Route Planning
in Road Networks

Peter Sanders **<u>Dominik Schultes</u>**

Institut für Theoretische Informatik – Algorithmik II

Universität Karlsruhe (TH)

in cooperation with

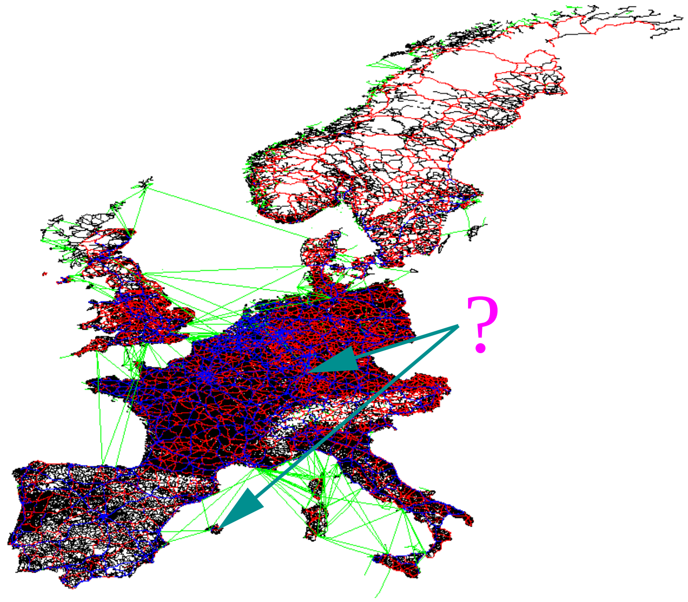
Holger Bast, Daniel Delling, Stefan Funke, Sebastian Knopp, Domagoj Matijevic,

Jens Maue, Frank Schulz, Dorothea Wagner

`http://algo2.iti.uka.de/schultes/hwy/`

Paris, June 20, 2007

Shortest Path Problem



- given a weighted, directed graph $G = (V, E)$ with
  - $n = |V|$ nodes,
  - $m = |E|$ edges
- given a source node $s \in V$ and target node $t \in V$
- task: determine the shortest path from s to t in G

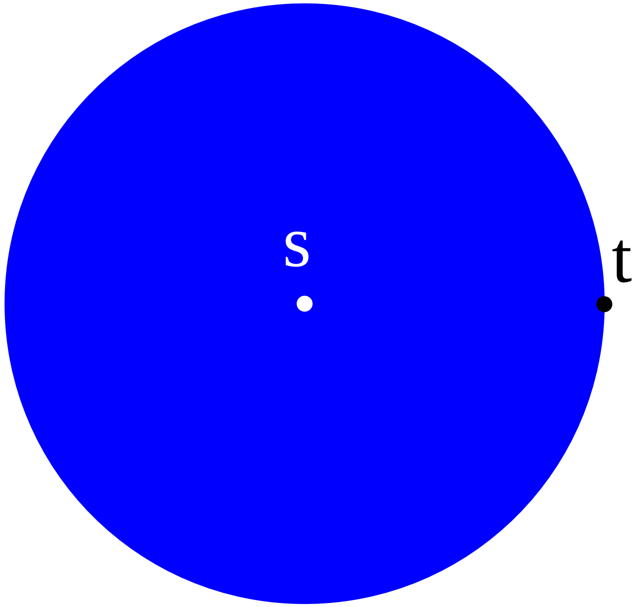
  (if there is any path from s to t)

DIJKSTRA's Algorithm

the classic solution [1959]

$O(n \log n + m)$ (with Fibonacci heaps)

Dijkstra



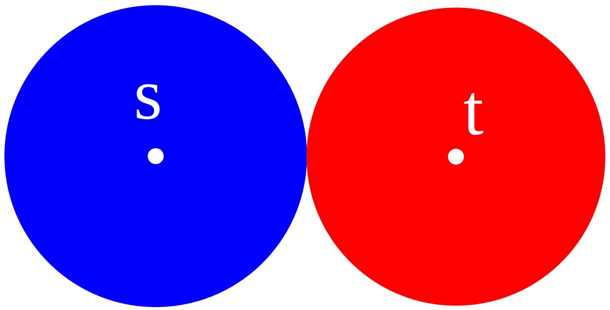
not practicable

for large graphs

(e.g. European road network:

$\approx$ 18 000 000 nodes)

bidirectional Dijkstra



improves the running time,

but still too slow

Speedup Techniques

that are faster than Dijkstra's algorithm

- require additional data (e.g., node coordinates)

  not always available!

AND / OR

- preprocess the graph and generate auxiliary data (e.g., 'signposts')

  can take a lot of time; assume static graph and many queries!

AND / OR

- exploit special properties of G (e.g., planar, hierarchical)

  fail when the given graph has not the desired properties!

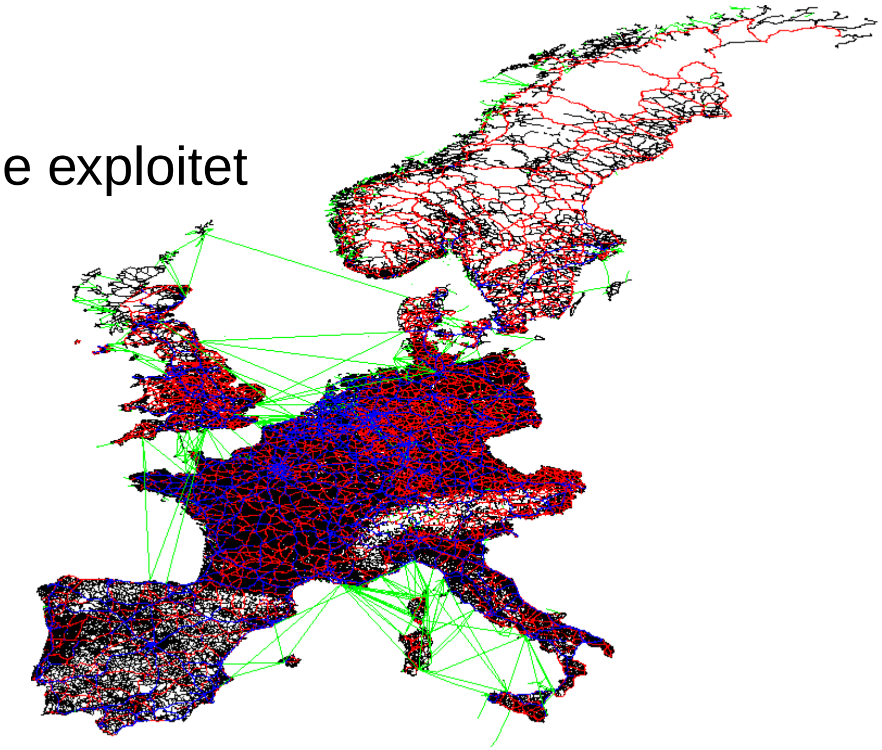
$\leadsto$ not a general solution,

but can be very efficient for many practically relevant cases

Road Networks

We concentrate on road networks.

- several useful properties that can be exploitet
- many real-world applications



Road Networks

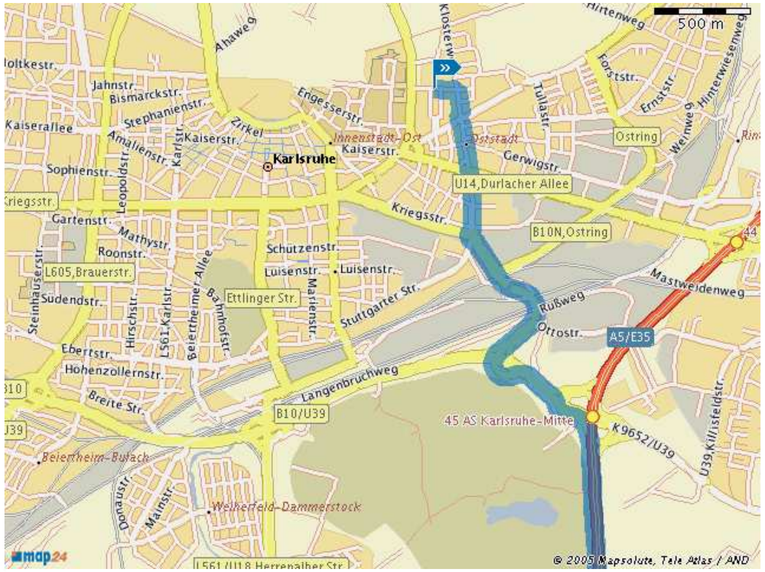
Properties

- large, e.g. $n = 18\,000\,000$ nodes for Western Europe
- sparse, i.e., $m = \Theta(n)$ edges
- almost planar, i.e., few edges cross
- inherent hierarchy, quickest paths use important streets
- changes are slow/few (only partly true!)

Road Networks

Applications

- route planning systems in the internet (e.g. `www.map24.de`)



- car navigation systems

- logistics planning
- traffic simulation

Outline

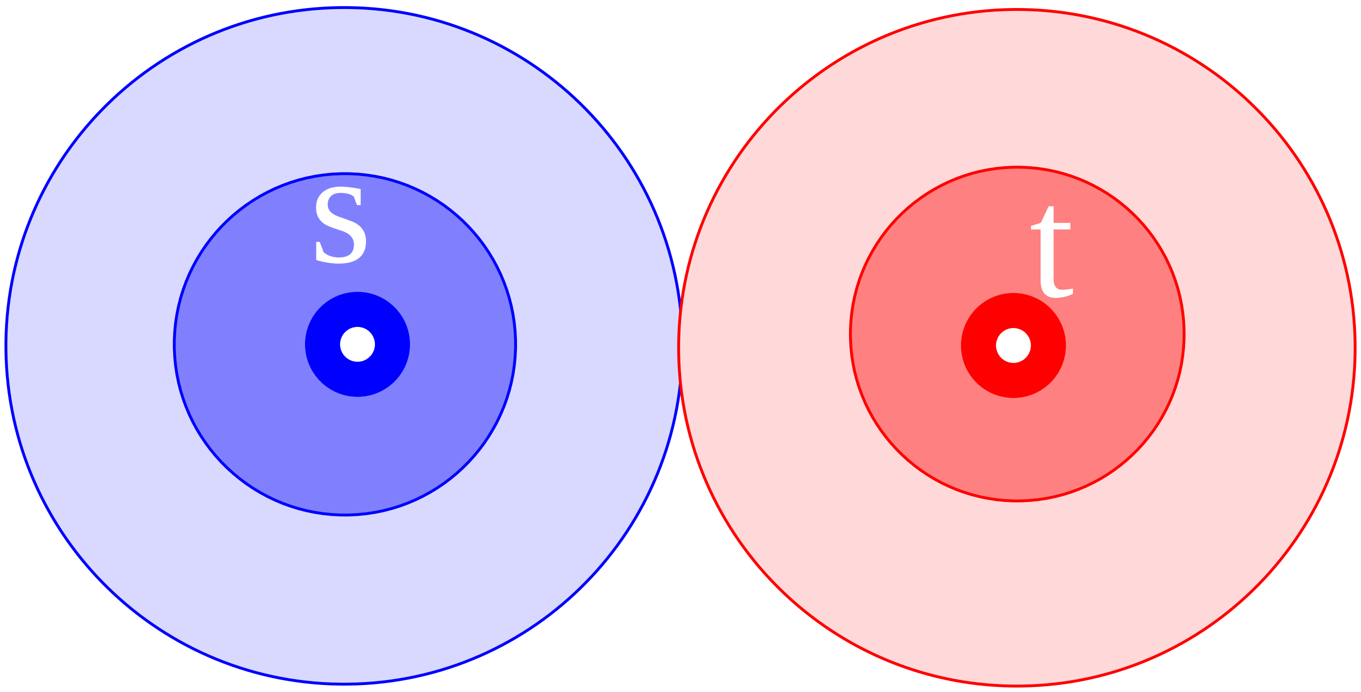
three different route planning approaches:

- **highway hierarchies** fast queries
- **transit-node routing** very fast queries
- **highway-node routing** very space-efficient, dynamic scenarios

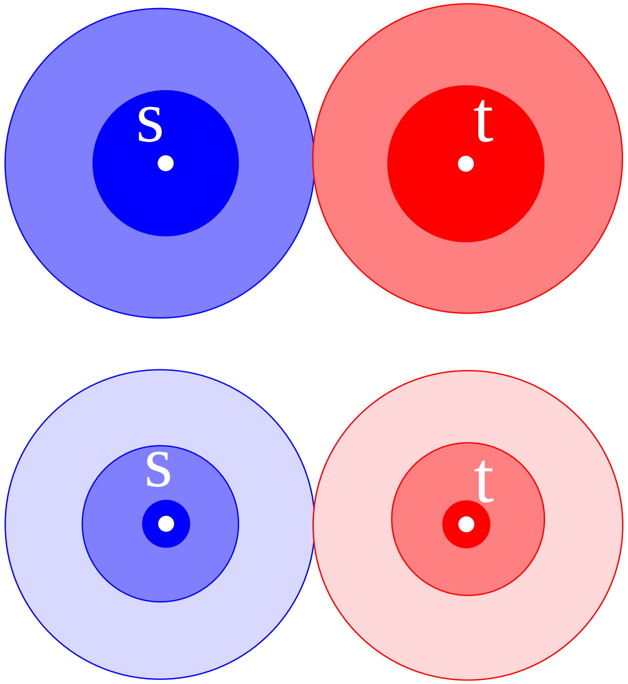
1. Approach

Highway Hierarchies

[SS 05–]



Commercial Approach



Heuristic Highway Hierarchy

- complete search in local area
- search in (sparser) highway network
- iterate ⇝ highway hierarchy

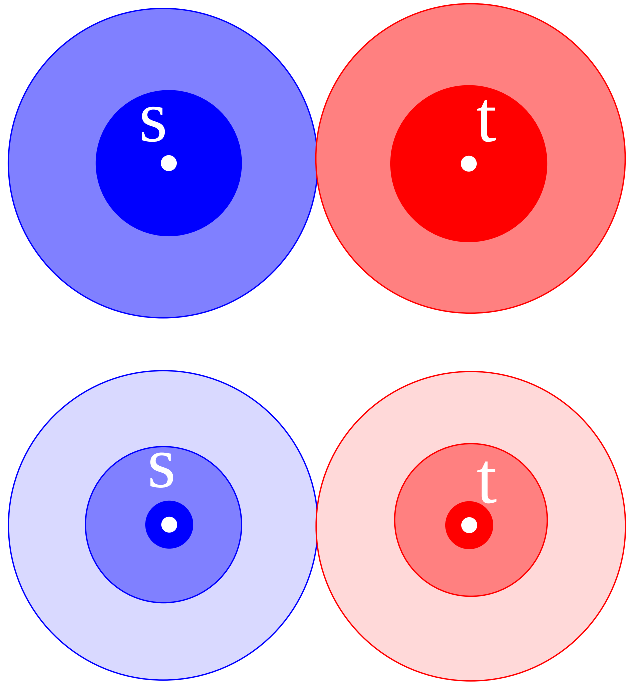
Defining the highway network:

use road category (highway, federal highway, motorway,. . .)

+ manual rectifications

- delicate compromise
- **speed ⇔ accuracy**

Our Approach



Exact Highway Hierarchy

- complete search in local area
- search in (sparser) highway network
- iterate ⇝ highway hierarchy

Defining the highway network:

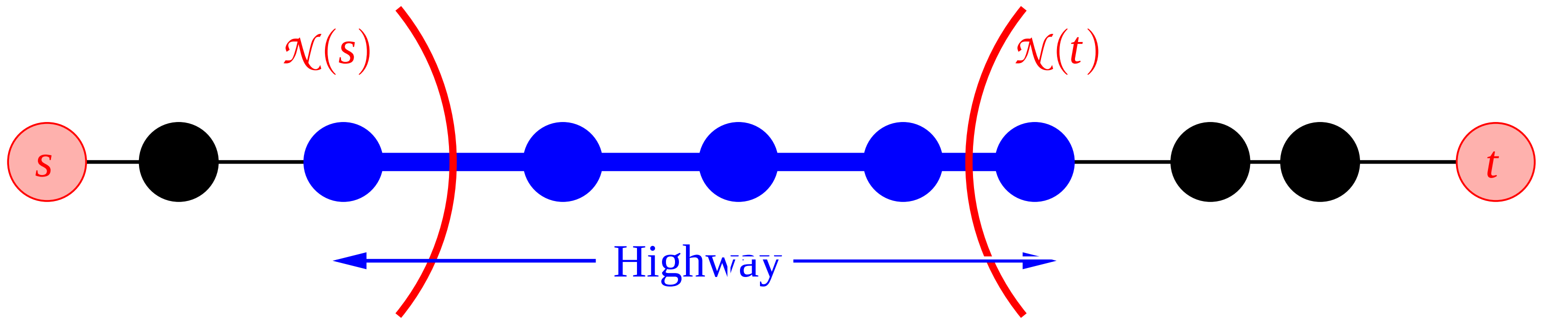
minimal network that preserves all shortest paths

- fully automatic (just fix neighborhood size)
- uncompromisingly **fast**

Constructing Exact Highway Hierarchies

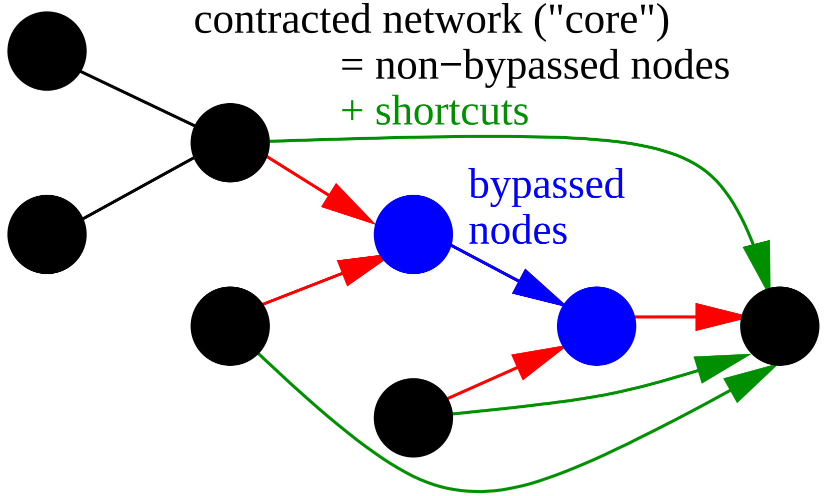
Alternate between two phases:

Edge reduction to highway edges needed outside local searches.

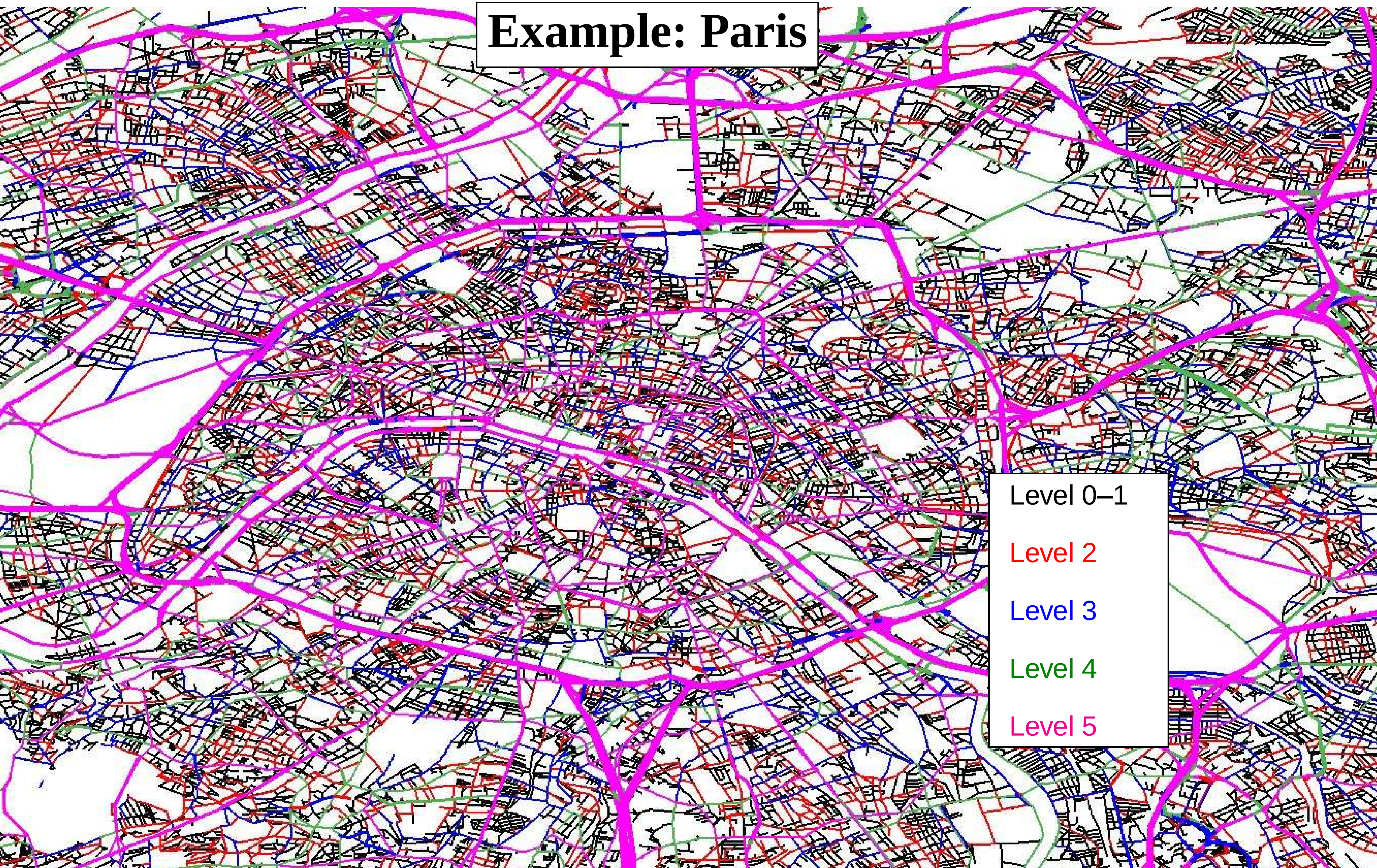


Node reduction.

Remove low degree nodes

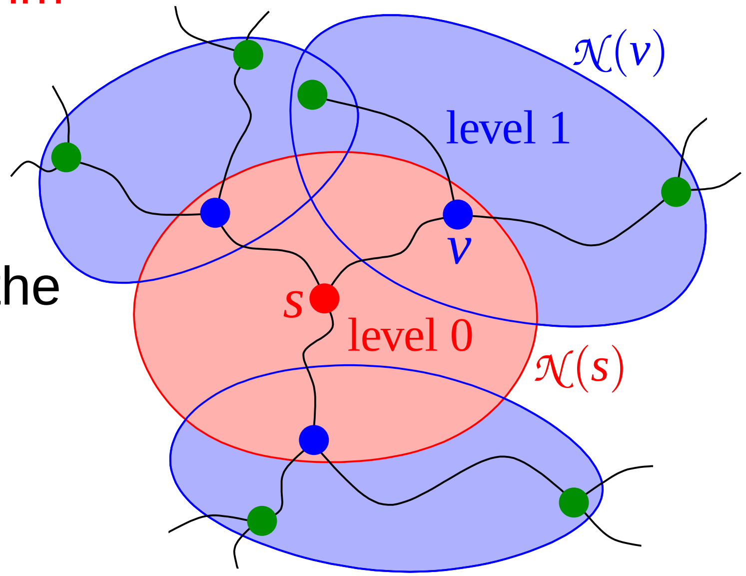


Example: Paris



Query

Bidirectional version of Dijkstra's Algorithm



Restrictions:

- Do not leave the neighbourhood of the entrance point to the current level.

  Instead: switch to the next level.

- Do not enter a component of bypassed nodes.

- entrance point to level 0
- entrance point to level 1
- entrance point to level 2

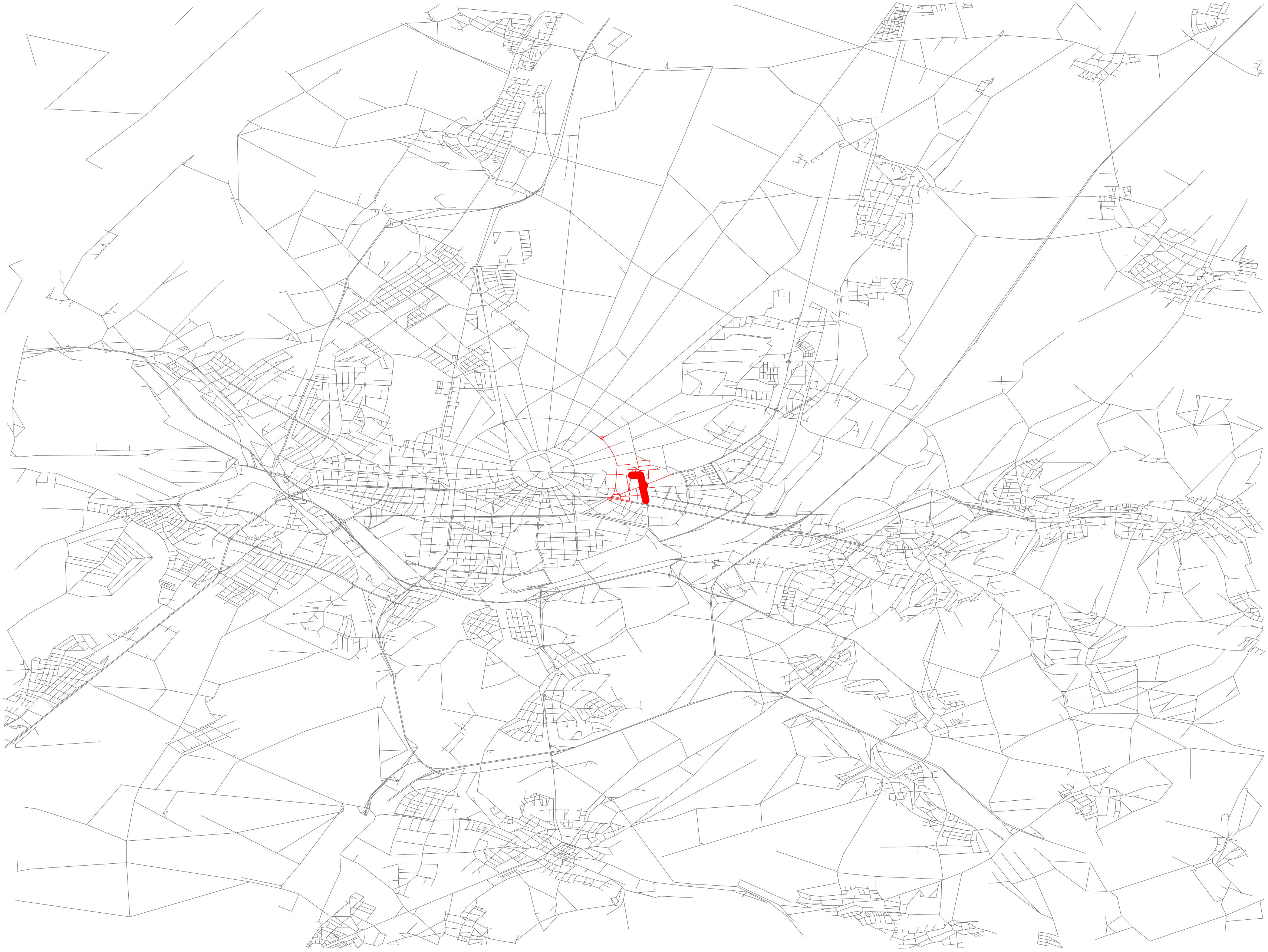
Query

Example: from Karlsruhe, Am Fasanengarten 5

to Palma de Mallorca

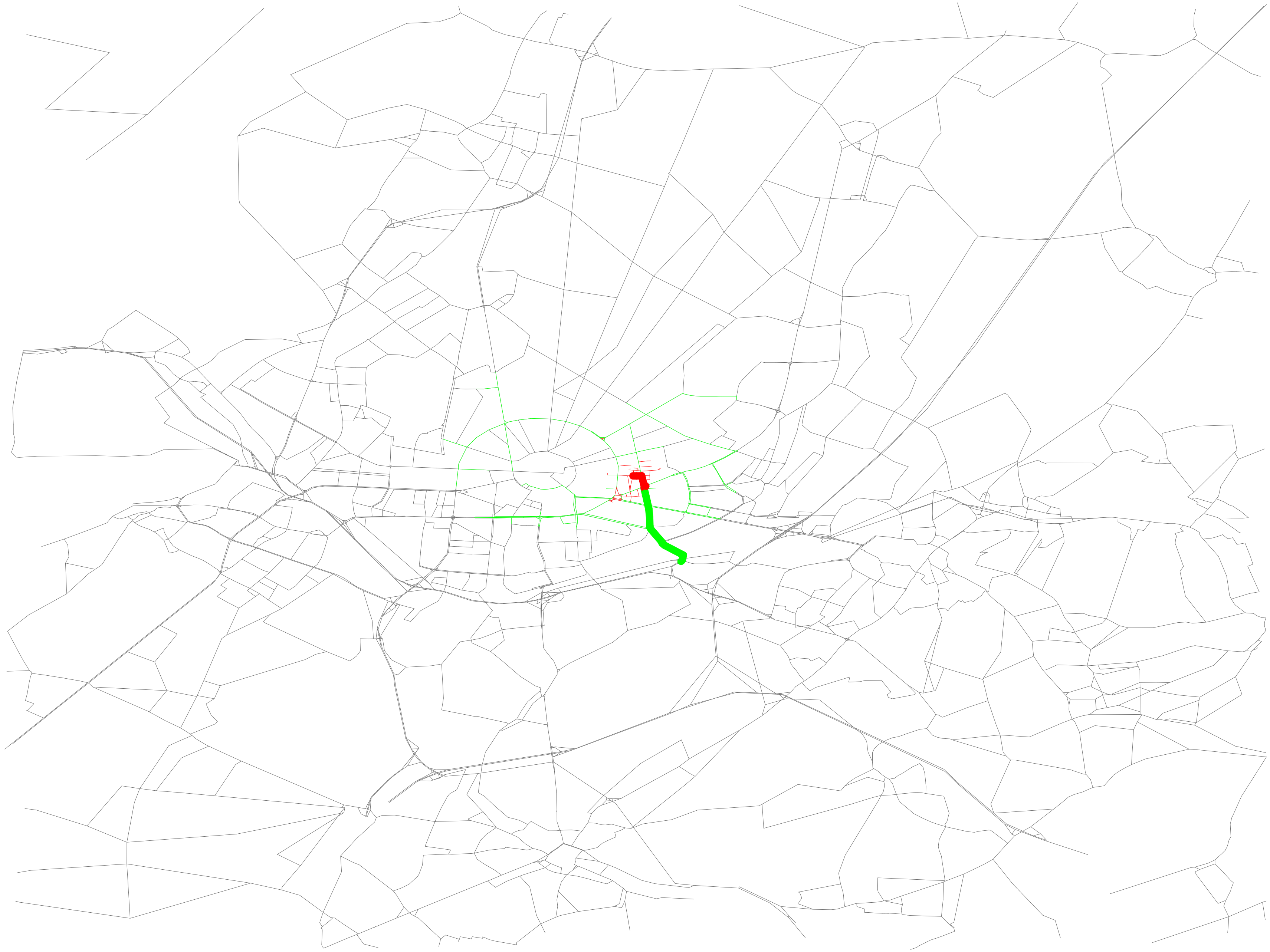
Bounding Box: 20 km Level 0

Bounding Box: 20 km Level 0 Search Space

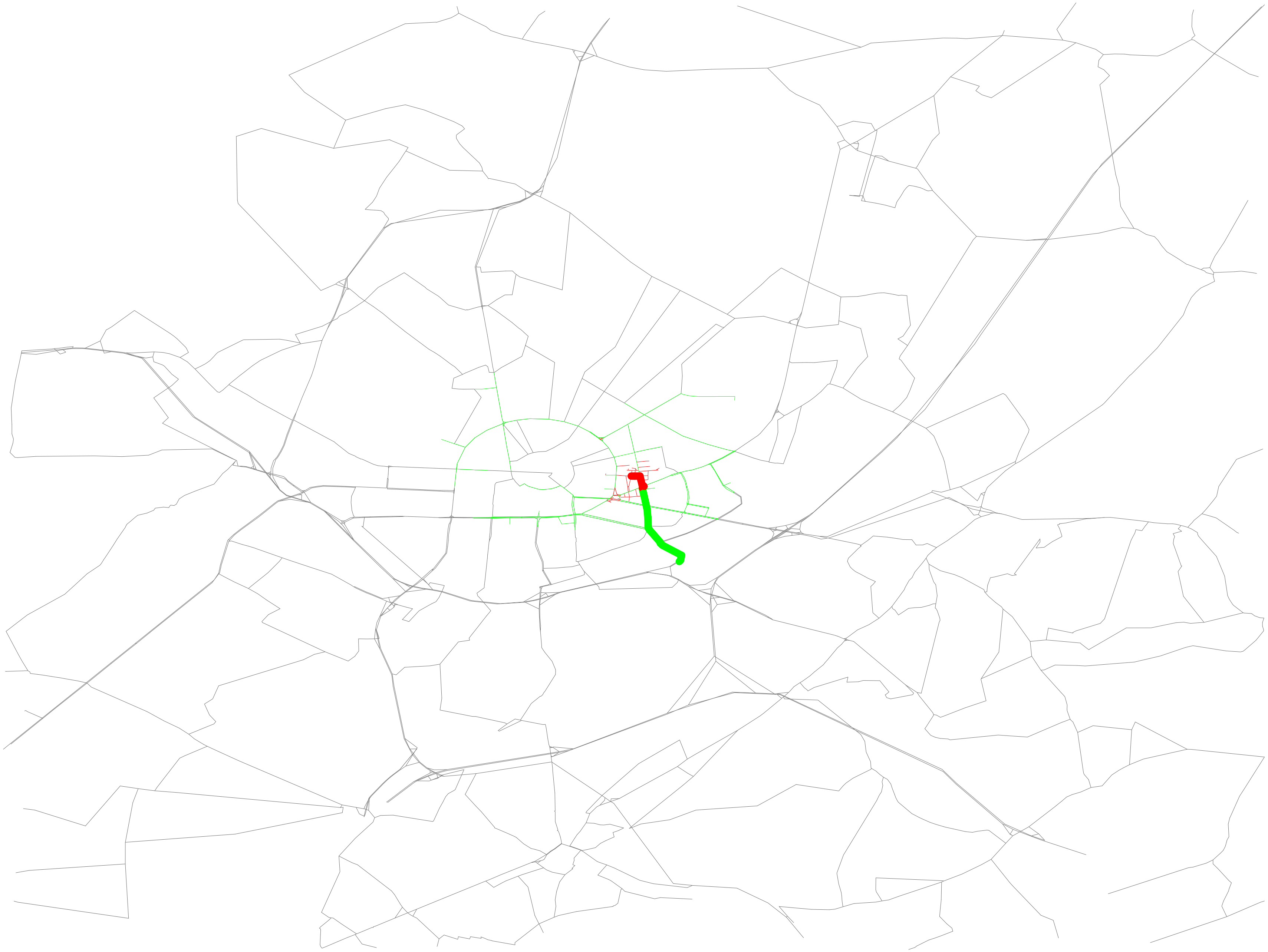


Bounding Box: 20 km Level 1

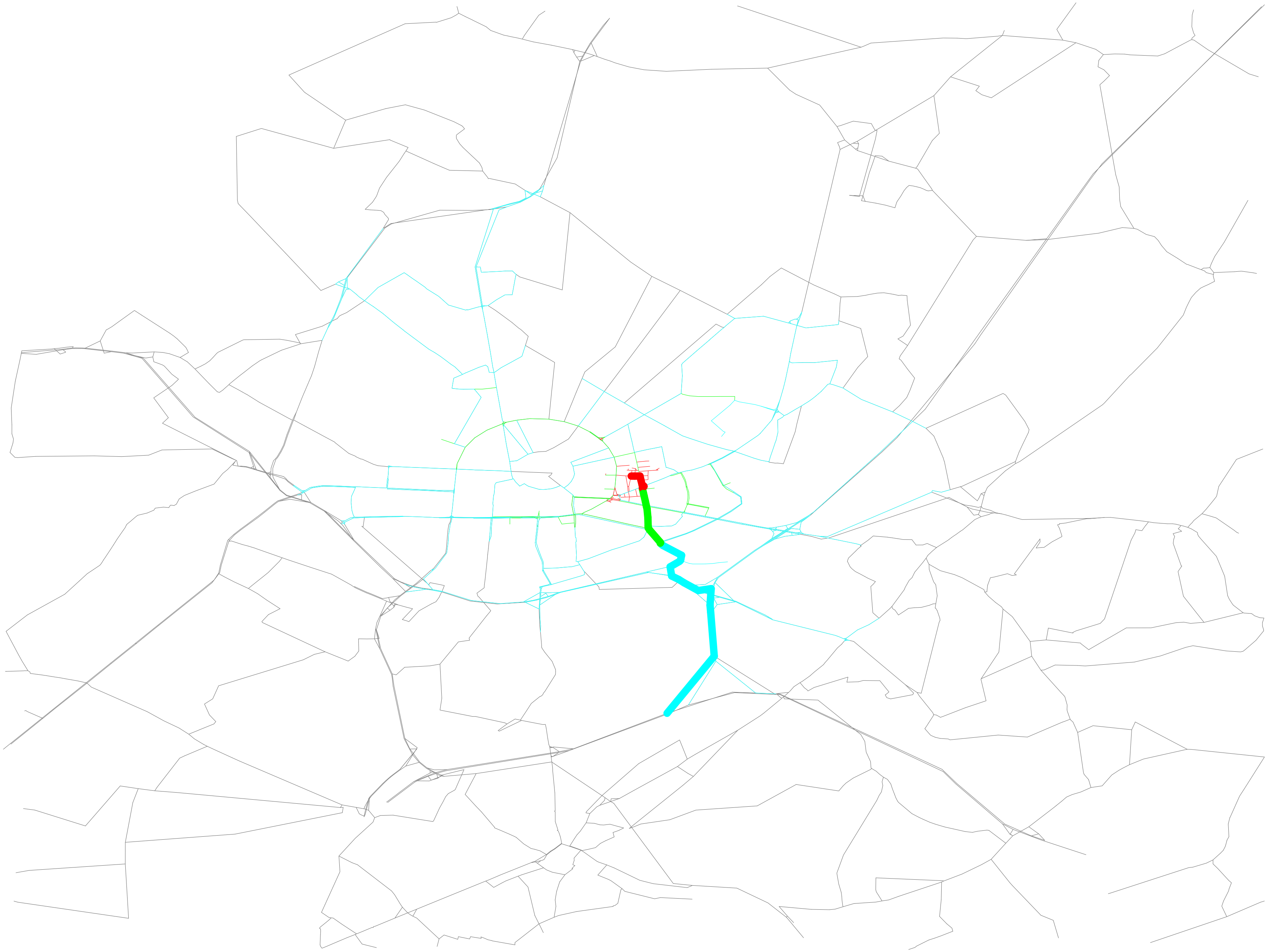
Bounding Box: 20 km Level 1 Search Space



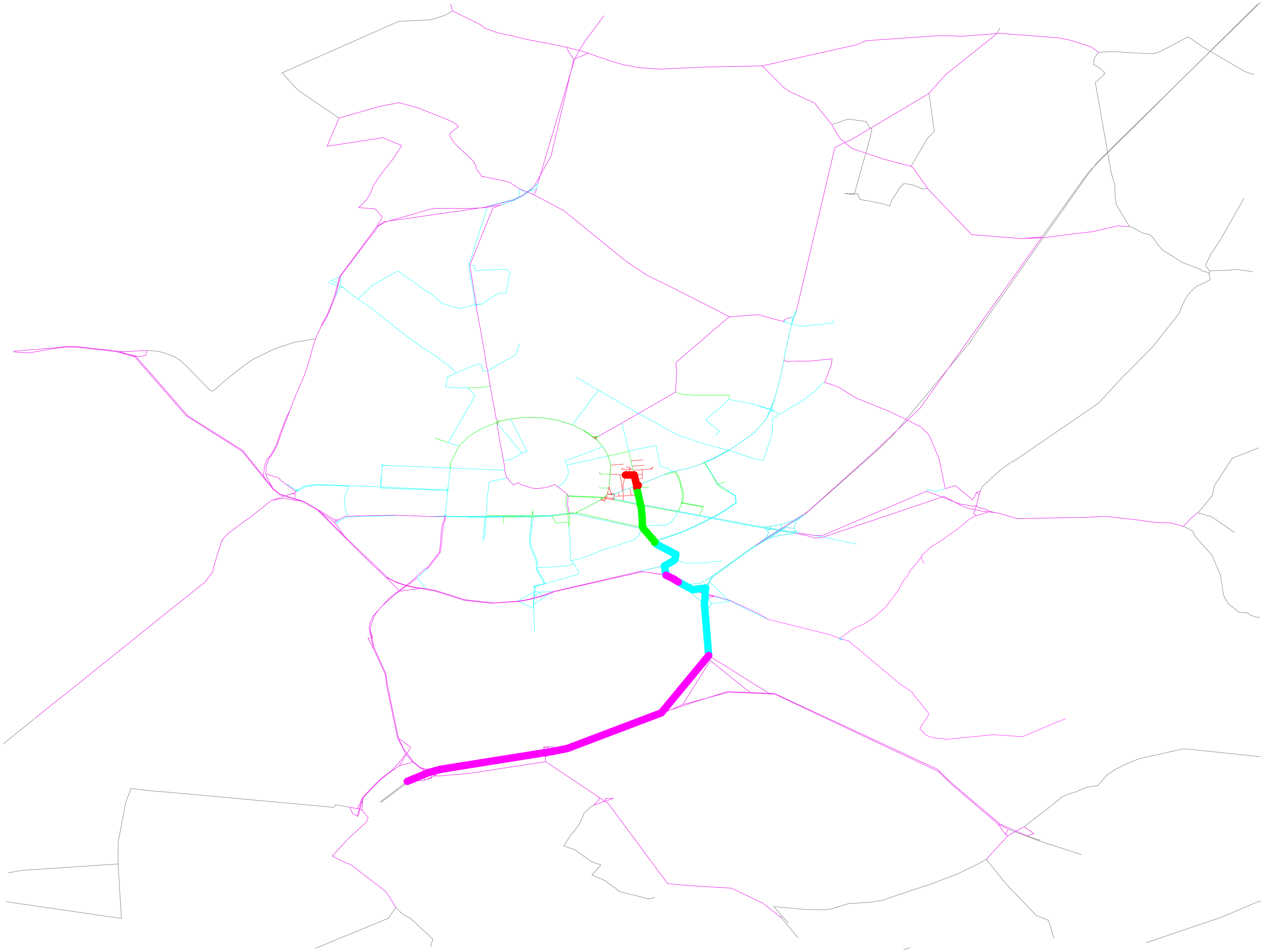
Bounding Box: 20 km Level 2



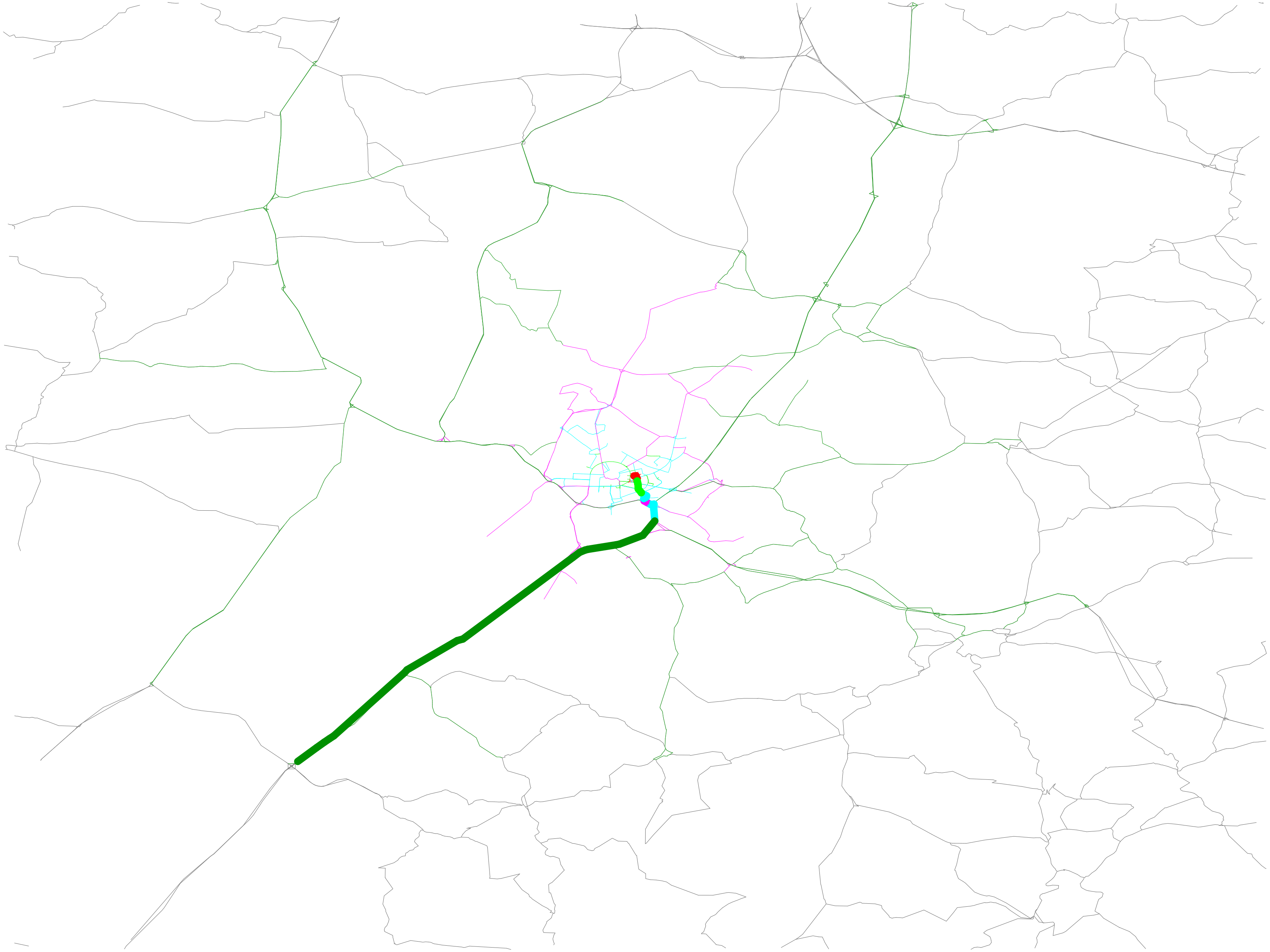
Bounding Box: 20 km Level 2 Search Space



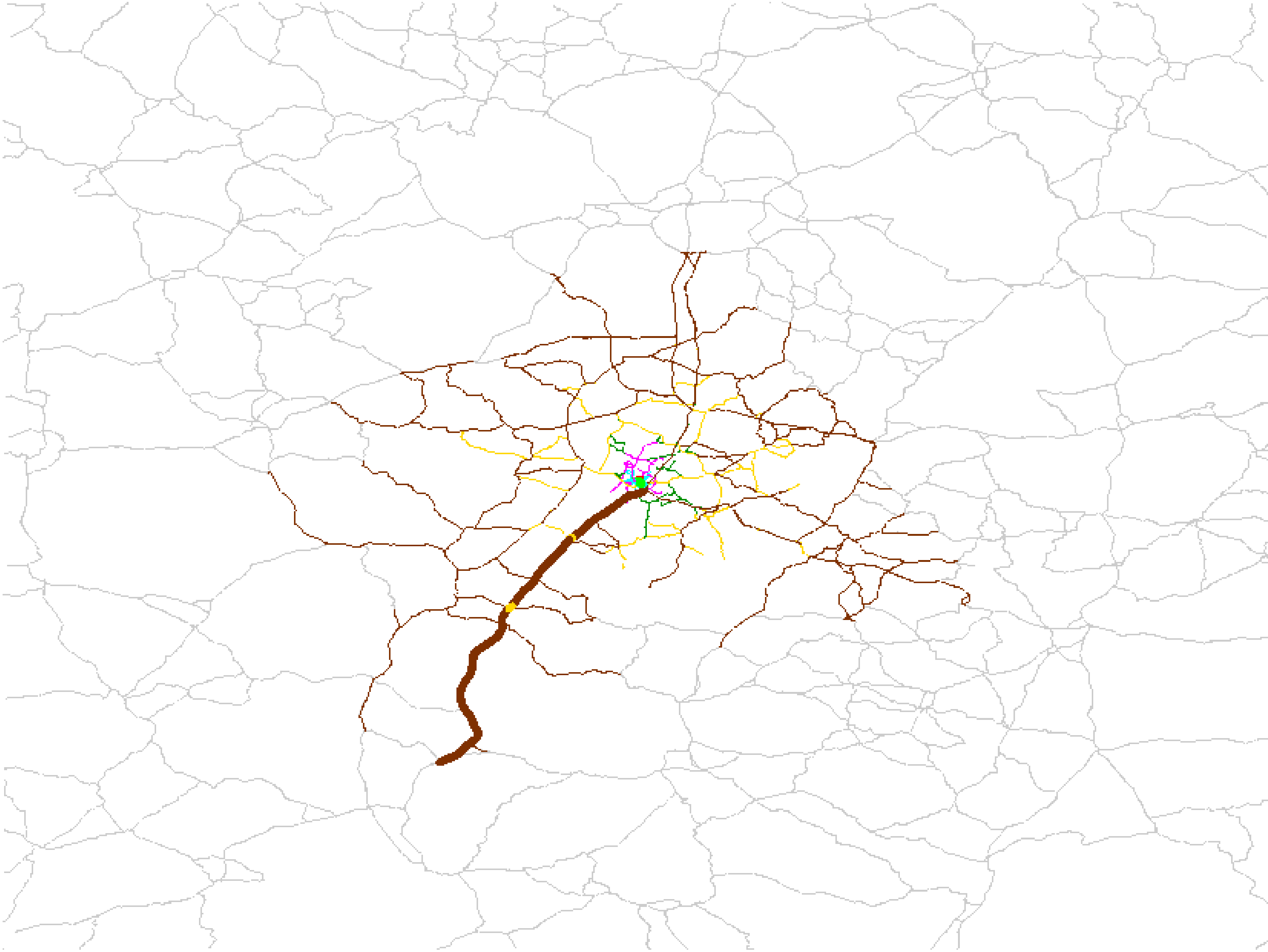
Bounding Box: 20 km Level 3 Search Space



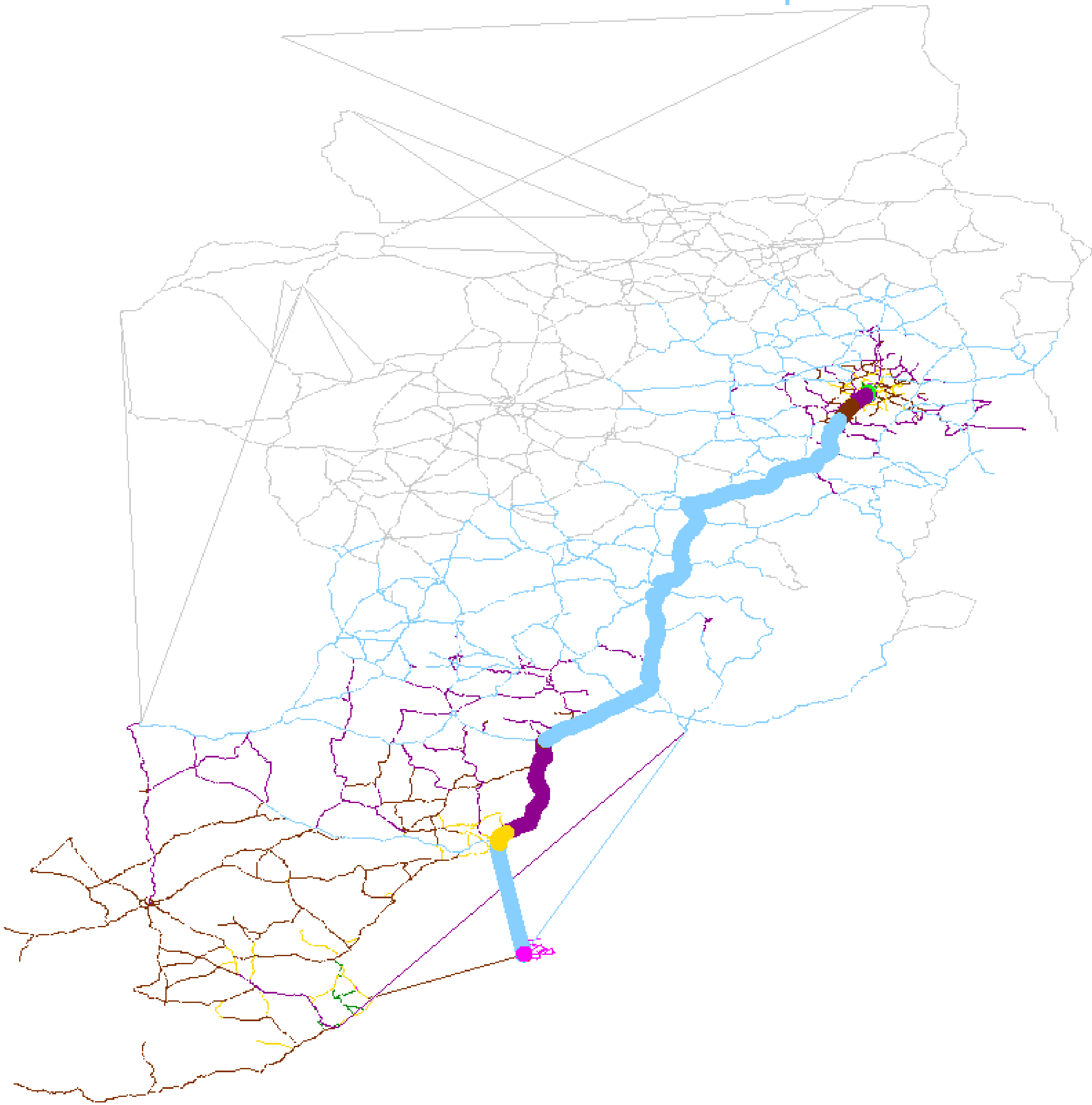
Bounding Box: 80 km Level 4 Search Space



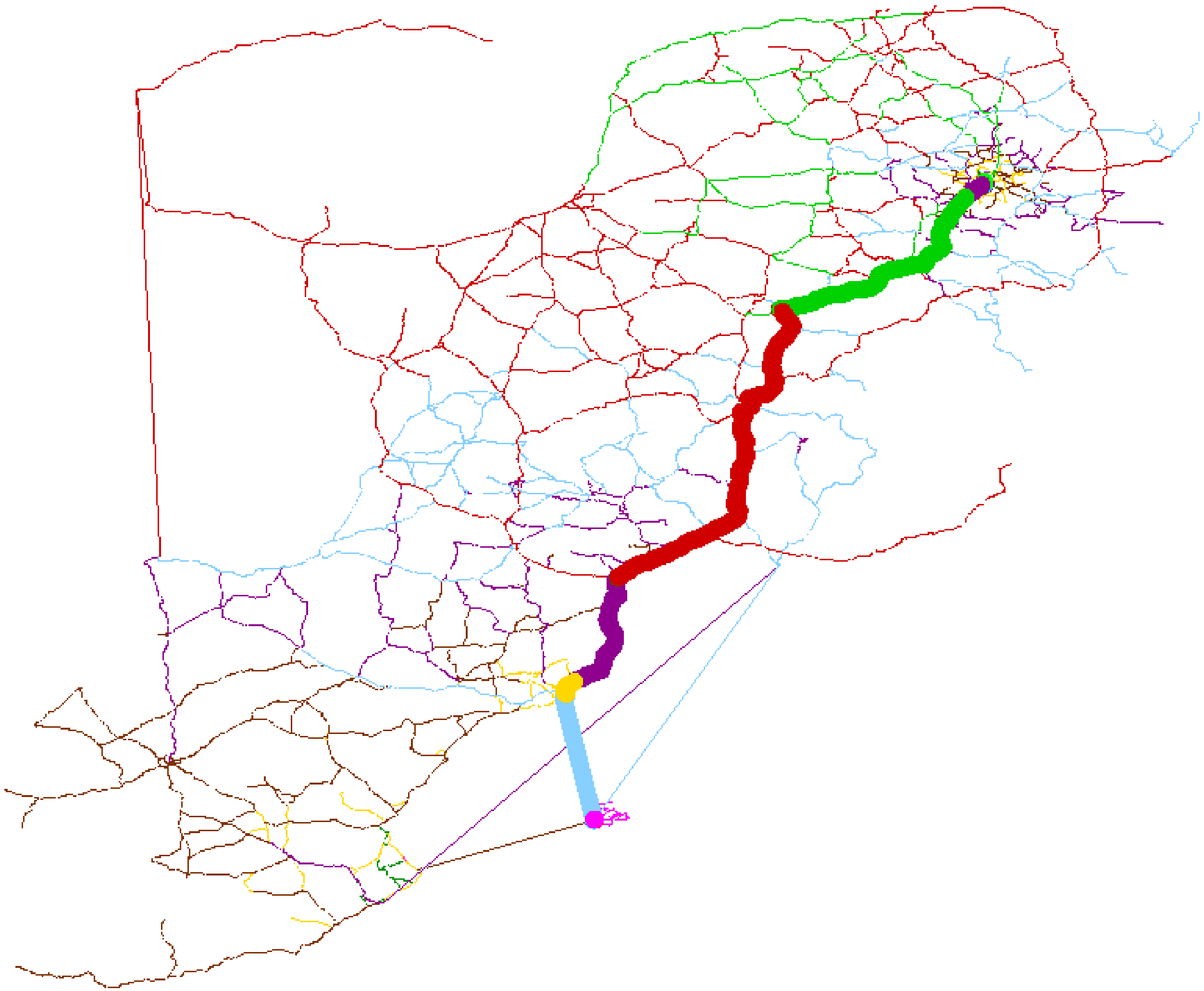
Bounding Box: 400 km Level 6 Search Space



Level 8 Search Space



Level 10 Search Space

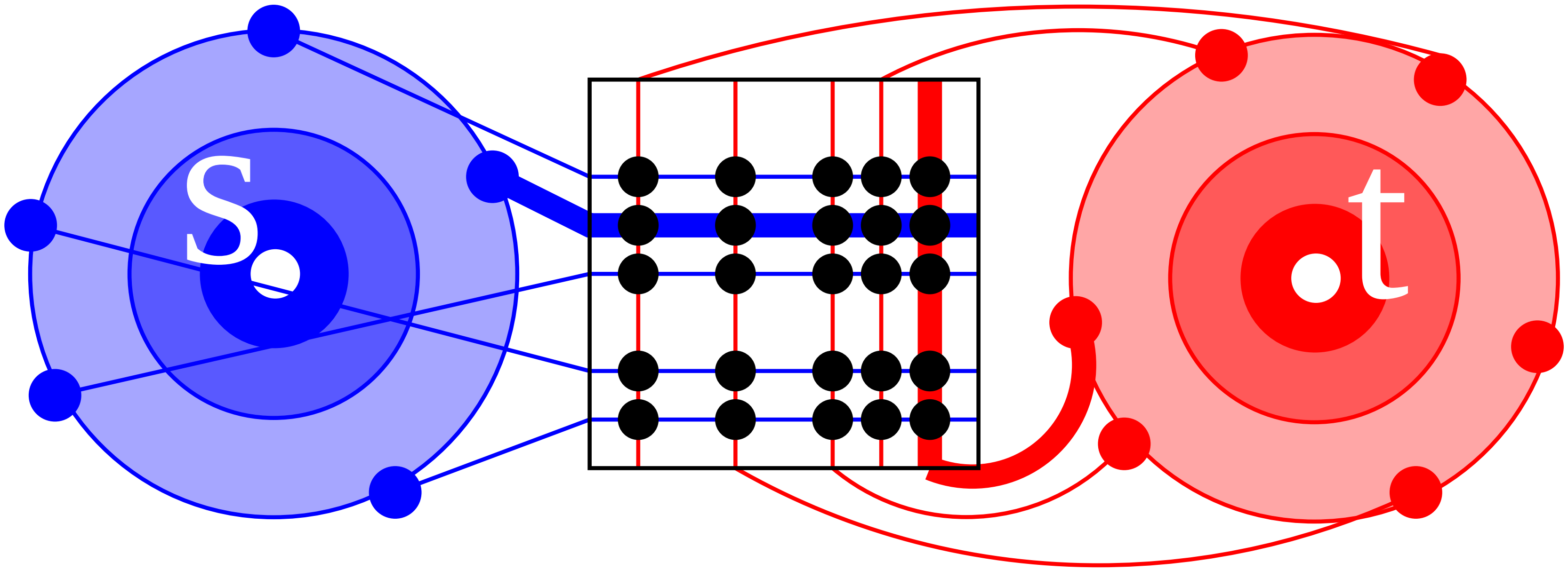


Optimisation: Distance Table

Construction:

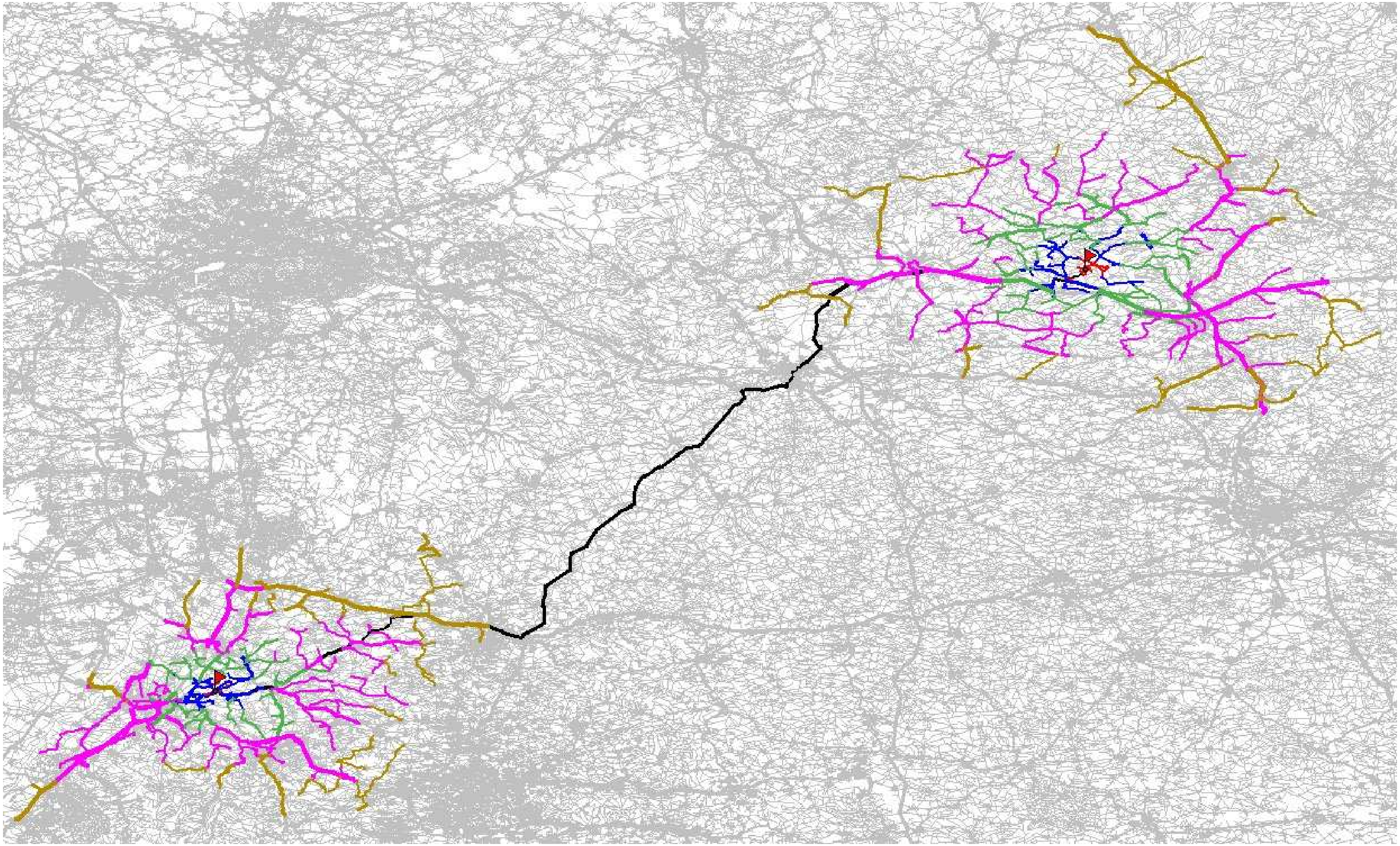
- Construct fewer levels. e.g. 4 instead of 9
- Compute an all-pairs distance table for the topmost level L. 13 465 $\times$ 13 465 entries

Distance Table Query

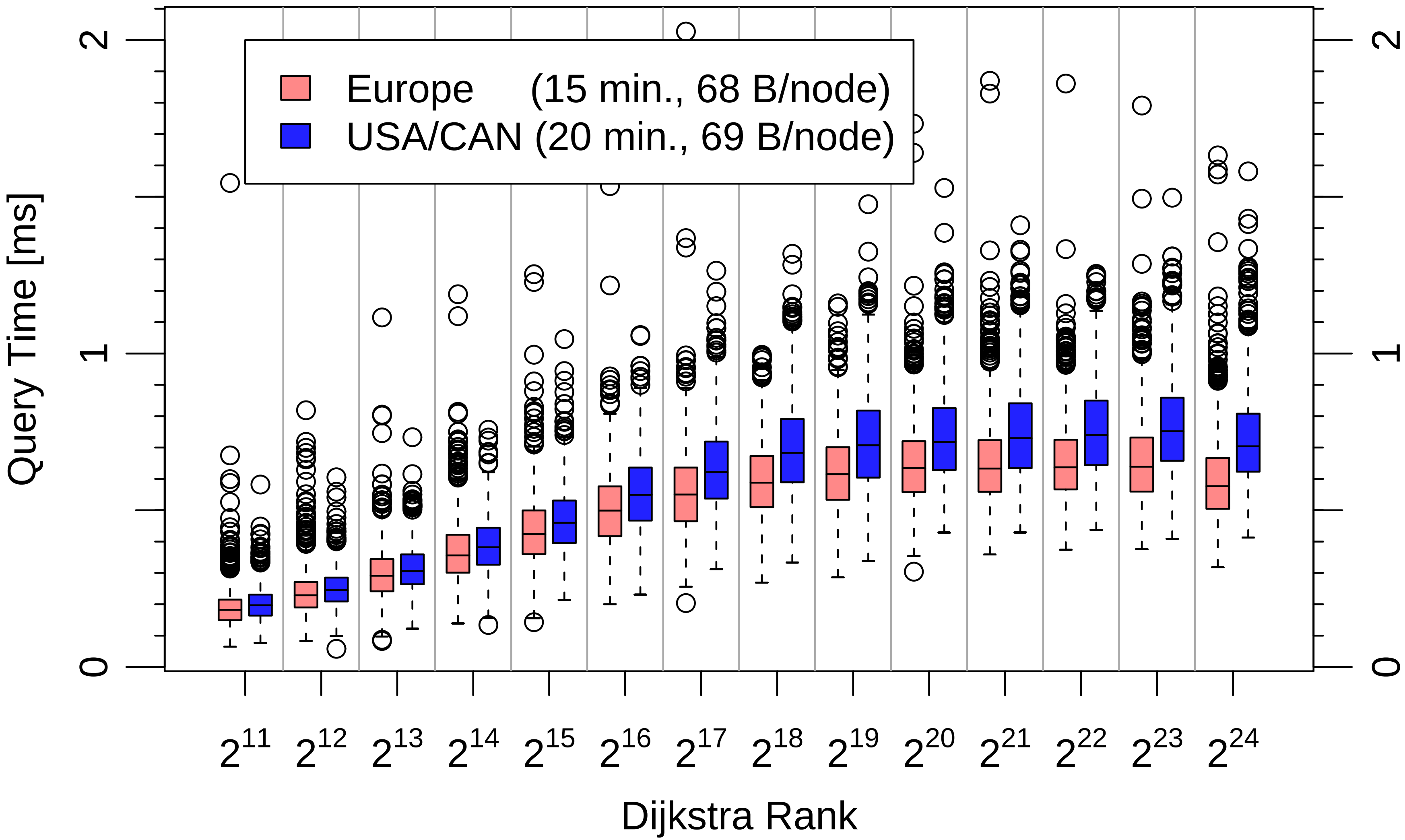


- Abort the search when all entrance points in the core of level L have been encountered. ≈ 55 for each direction
- Use the distance table to bridge the gap. $\approx 55 \times 55$ entries

Distance Table: Search Space Example

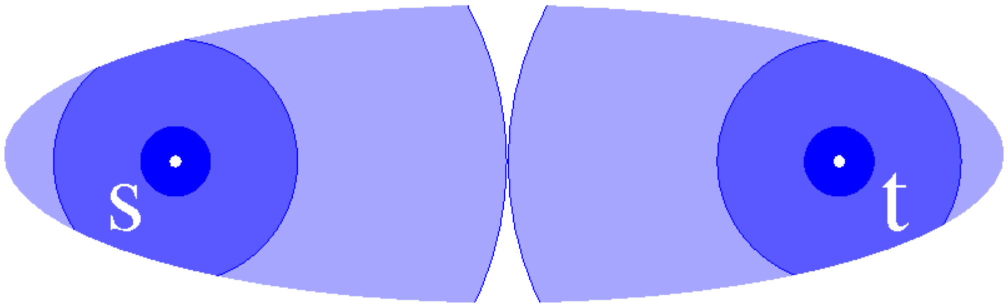


Local Queries (Highway Hierarchies)



Combination Goal Directed Search (landmarks)

[with D. Delling, D. Wagner]



- About 20 % faster than HHs + distance tables
- Significant speedup for approximate queries

Many-to-Many Routing

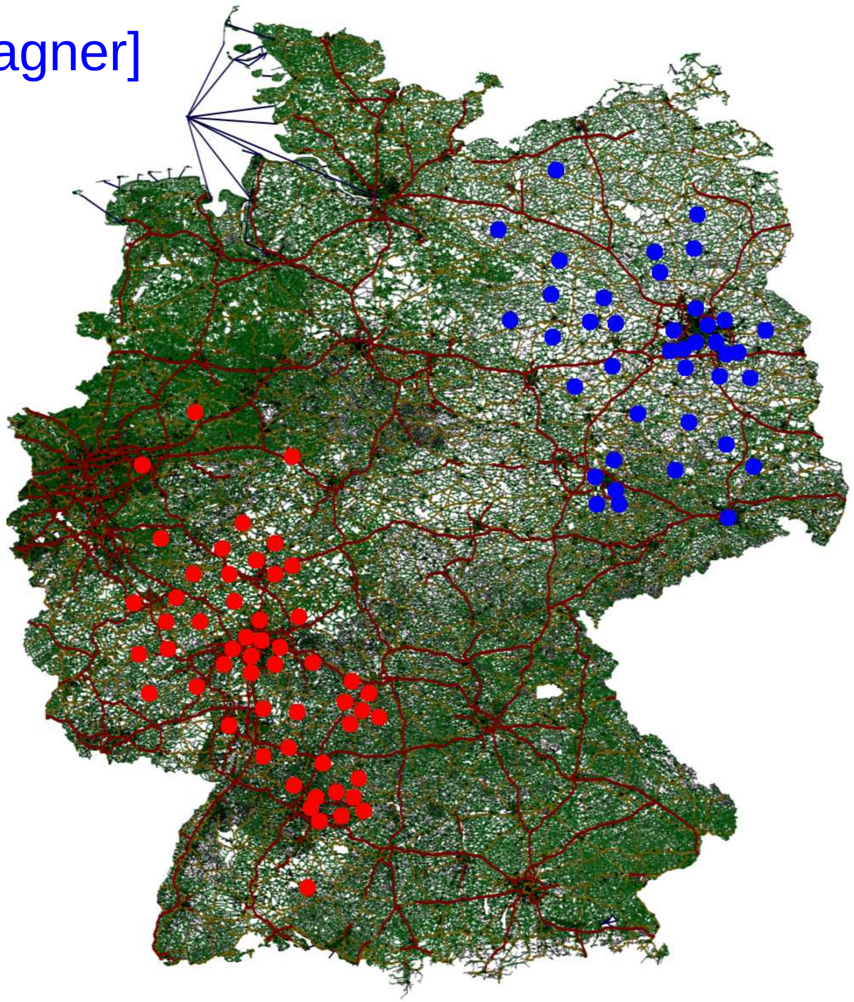
[with S. Knopp, F. Schulz (PTV AG), D. Wagner]

Find distances for all $(s,t) \in S \times T$

Applications: vehicle routing, TSP,

traffic simulation,

subroutine in preprocessing algorithms.



For example,

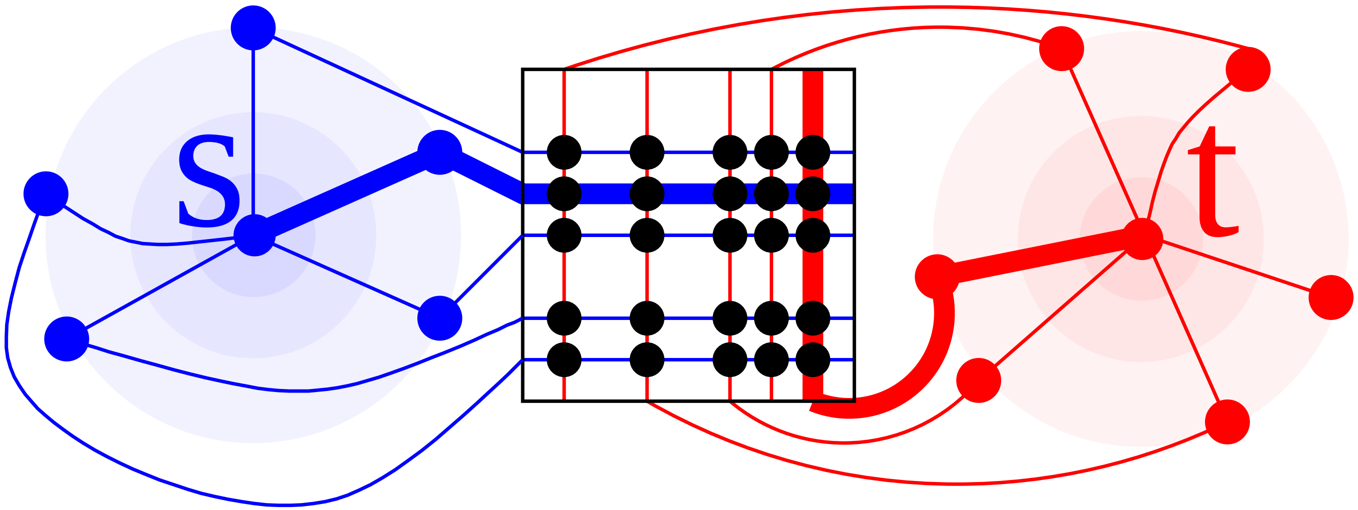
$10\,000 \times 10\,000$ table

in ≈ 1 min

2. Approach

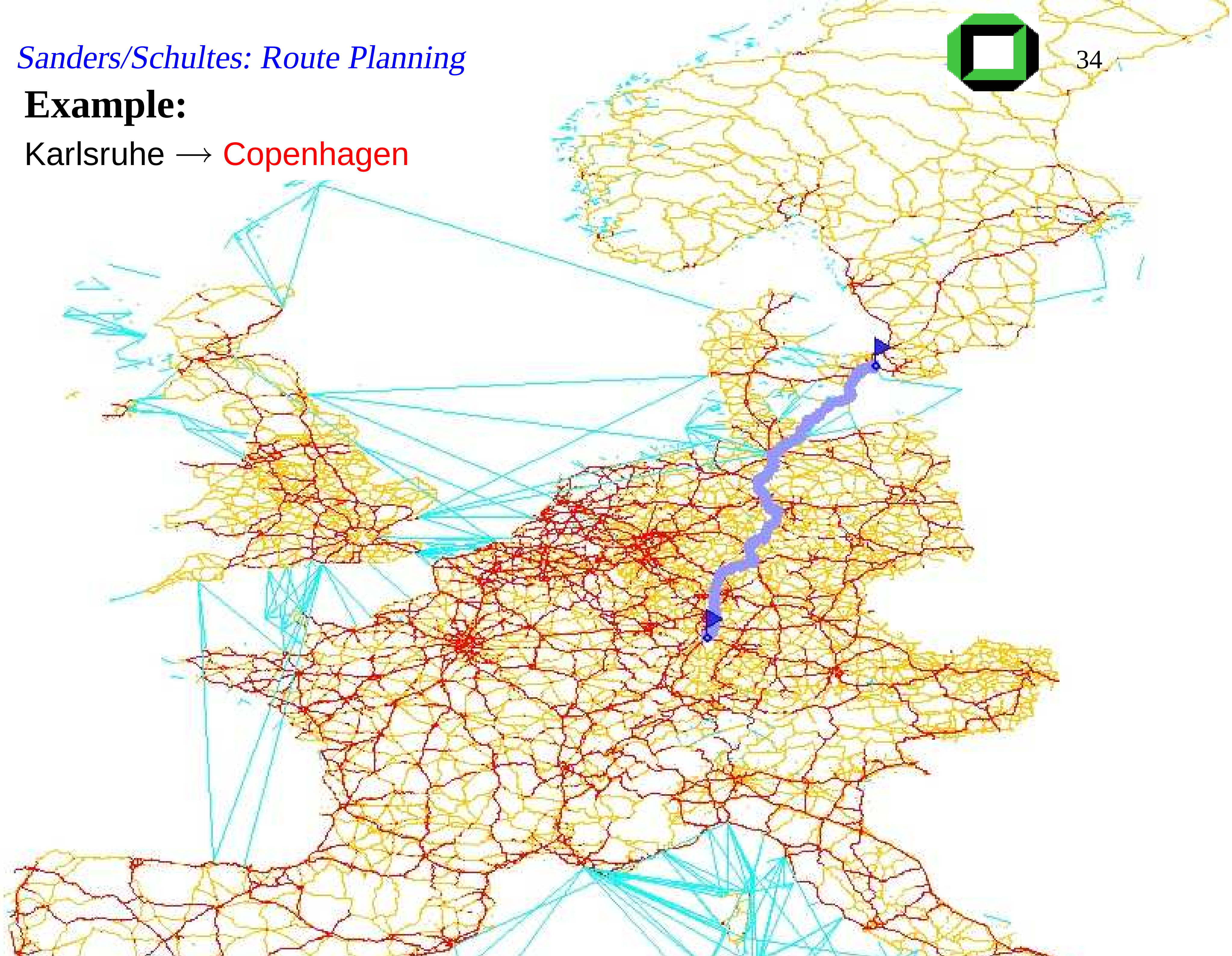
Transit-Node Routing

[with H. Bast and S. Funke]



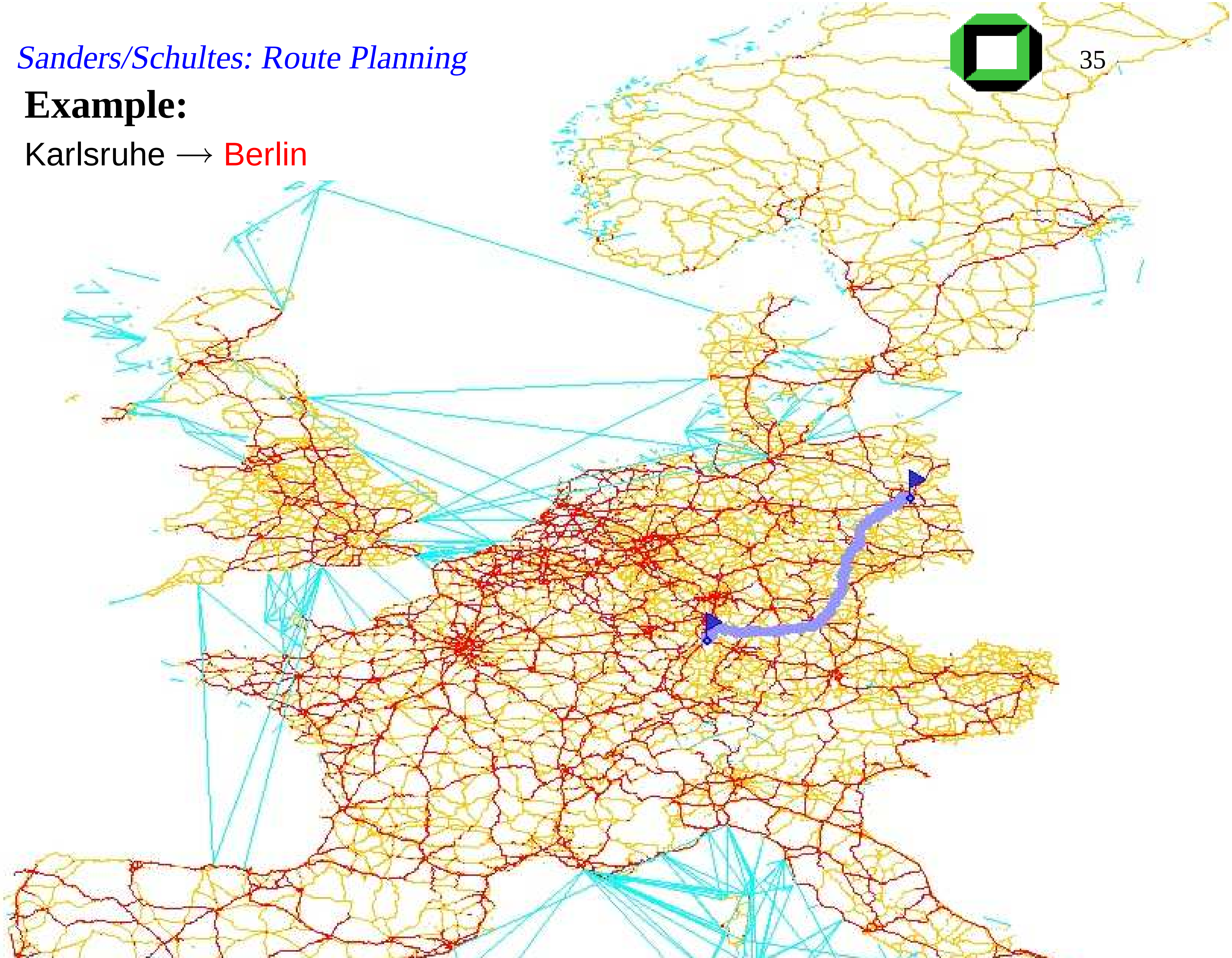
Example:

Karlsruhe $\longrightarrow$ Copenhagen



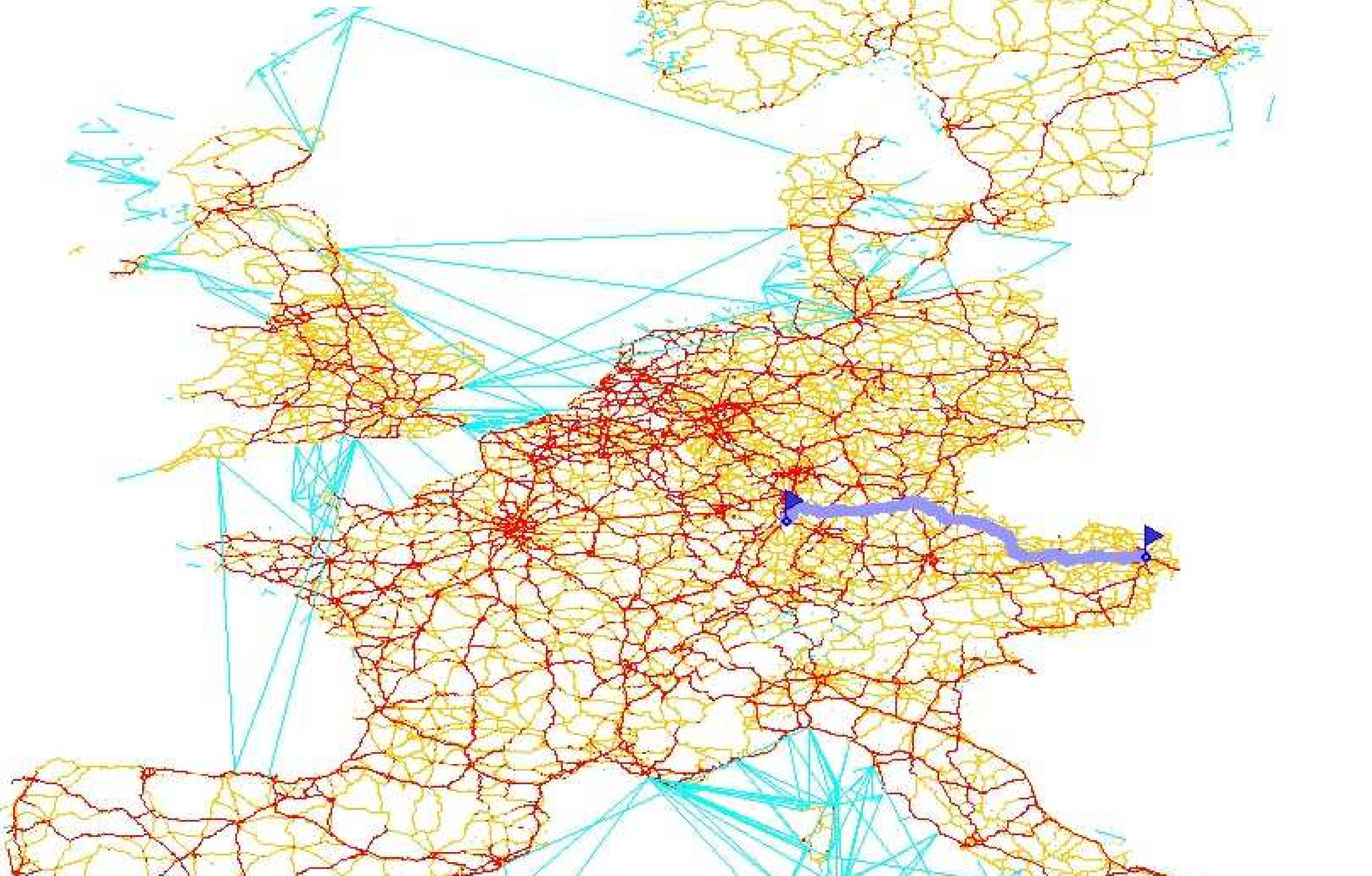
Example:

Karlsruhe $\longrightarrow$ Berlin



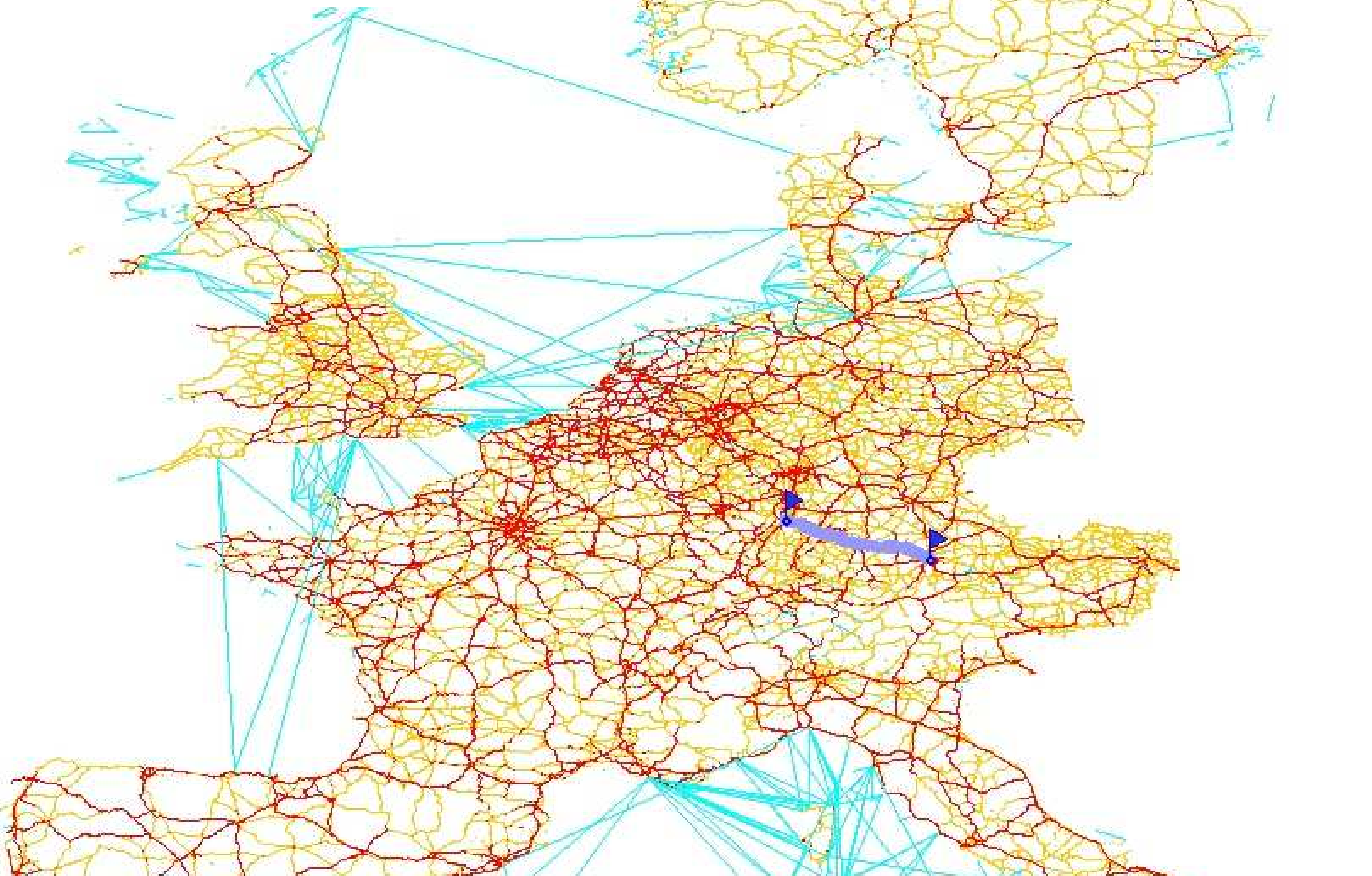
Example:

Karlsruhe $\longrightarrow$ Vienna



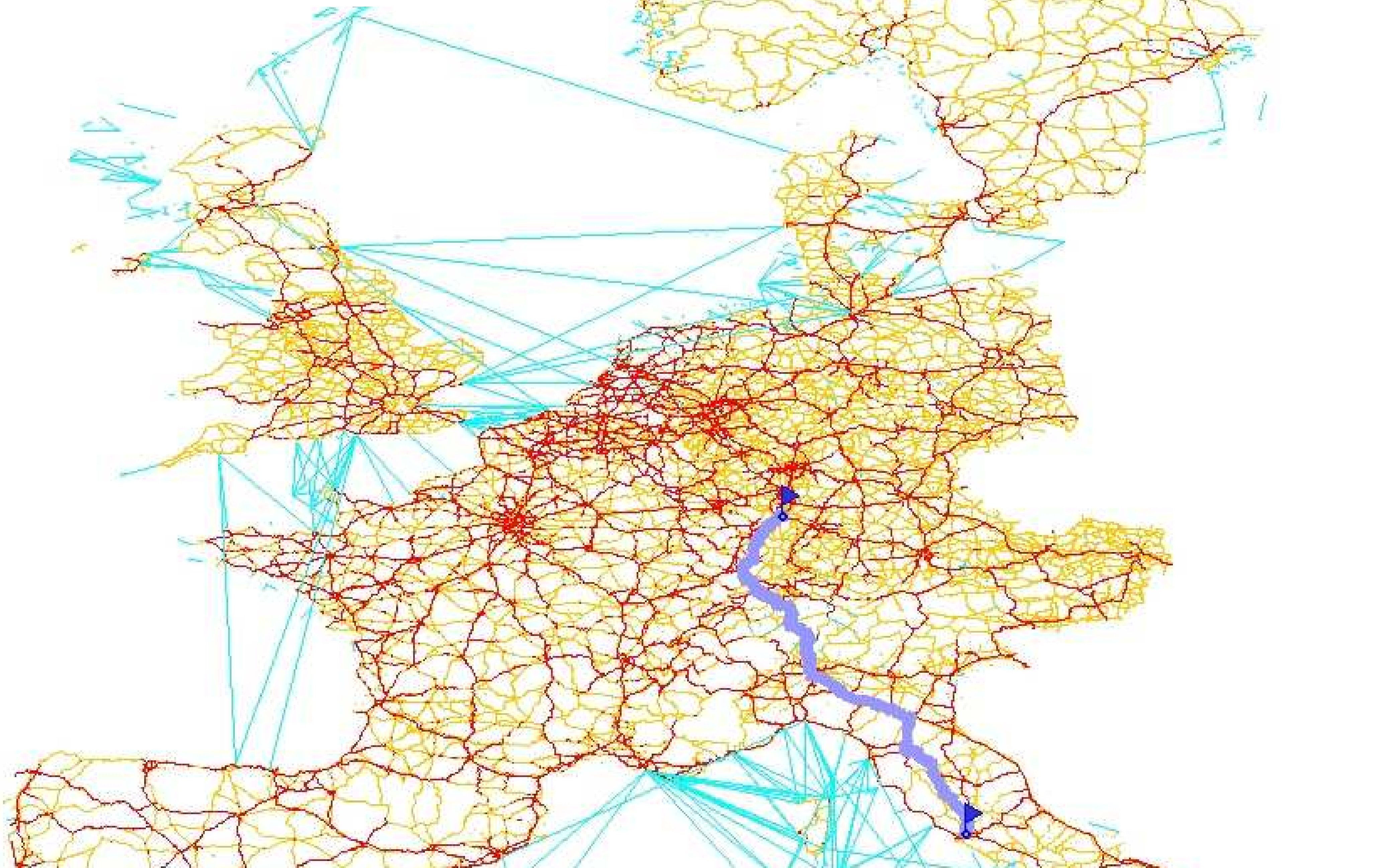
Example:

Karlsruhe $\longrightarrow$ Munich



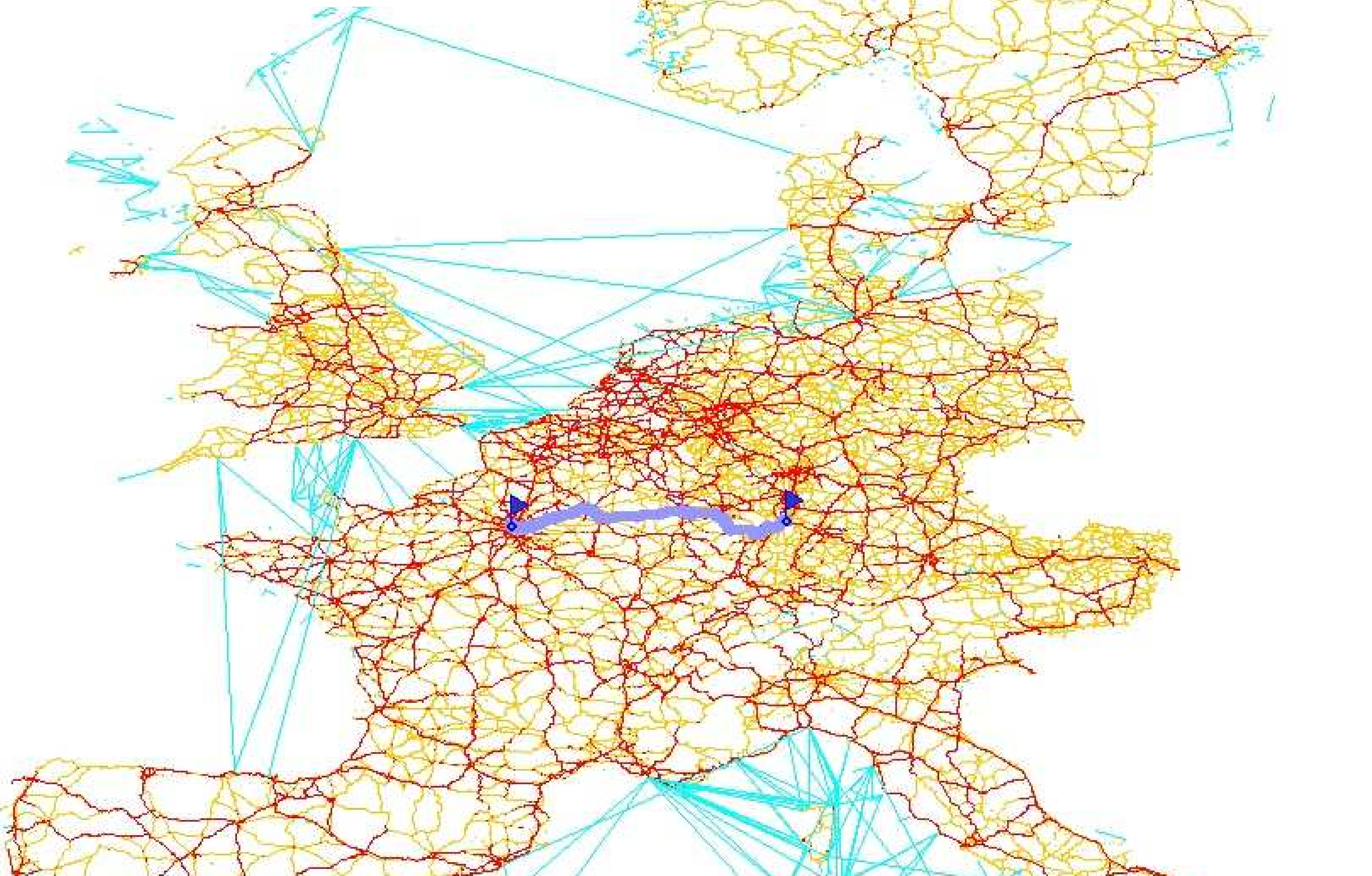
Example:

Karlsruhe → Rome



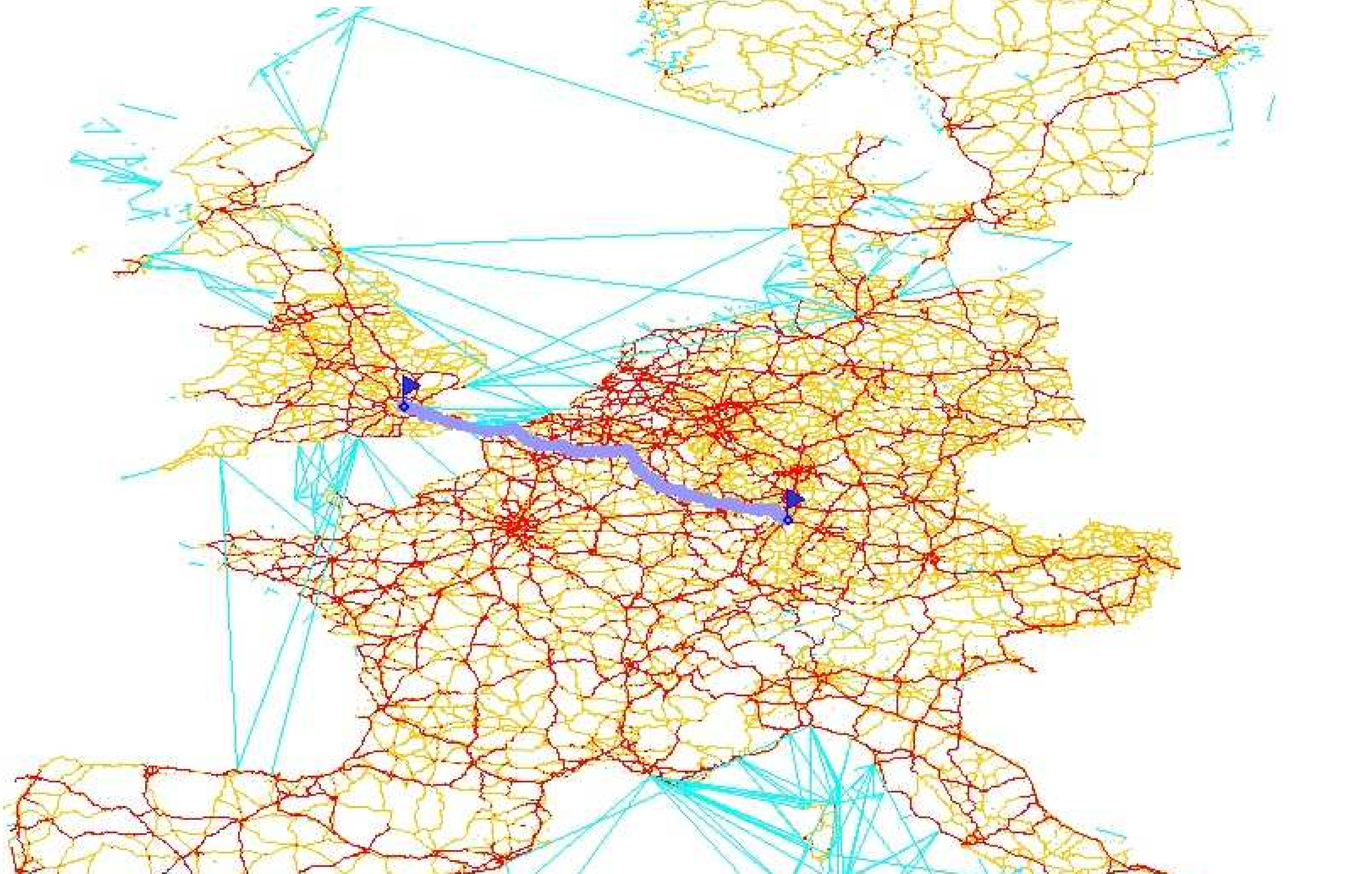
Example:

Karlsruhe $\longrightarrow$ Paris



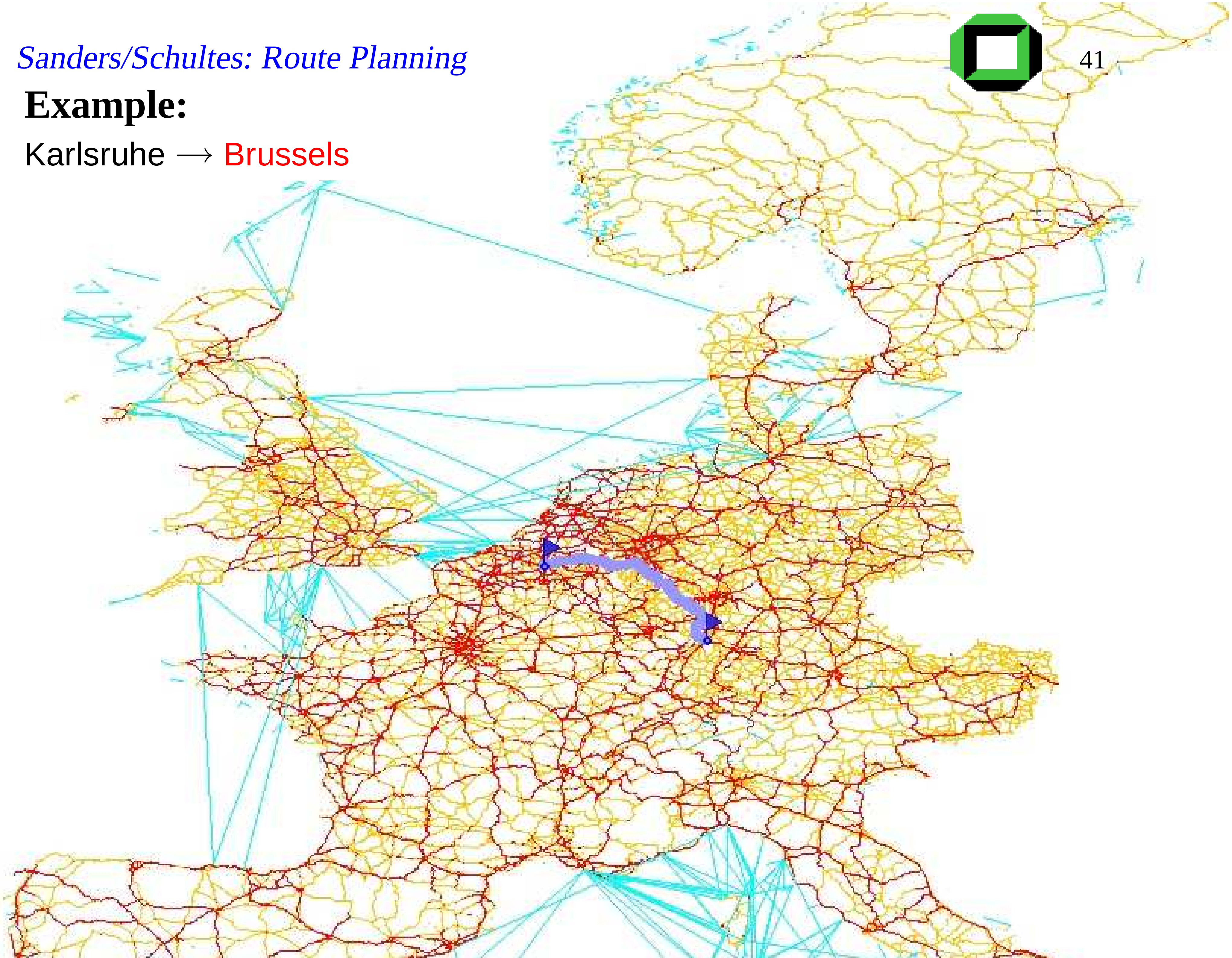
Example:

Karlsruhe $\longrightarrow$ London



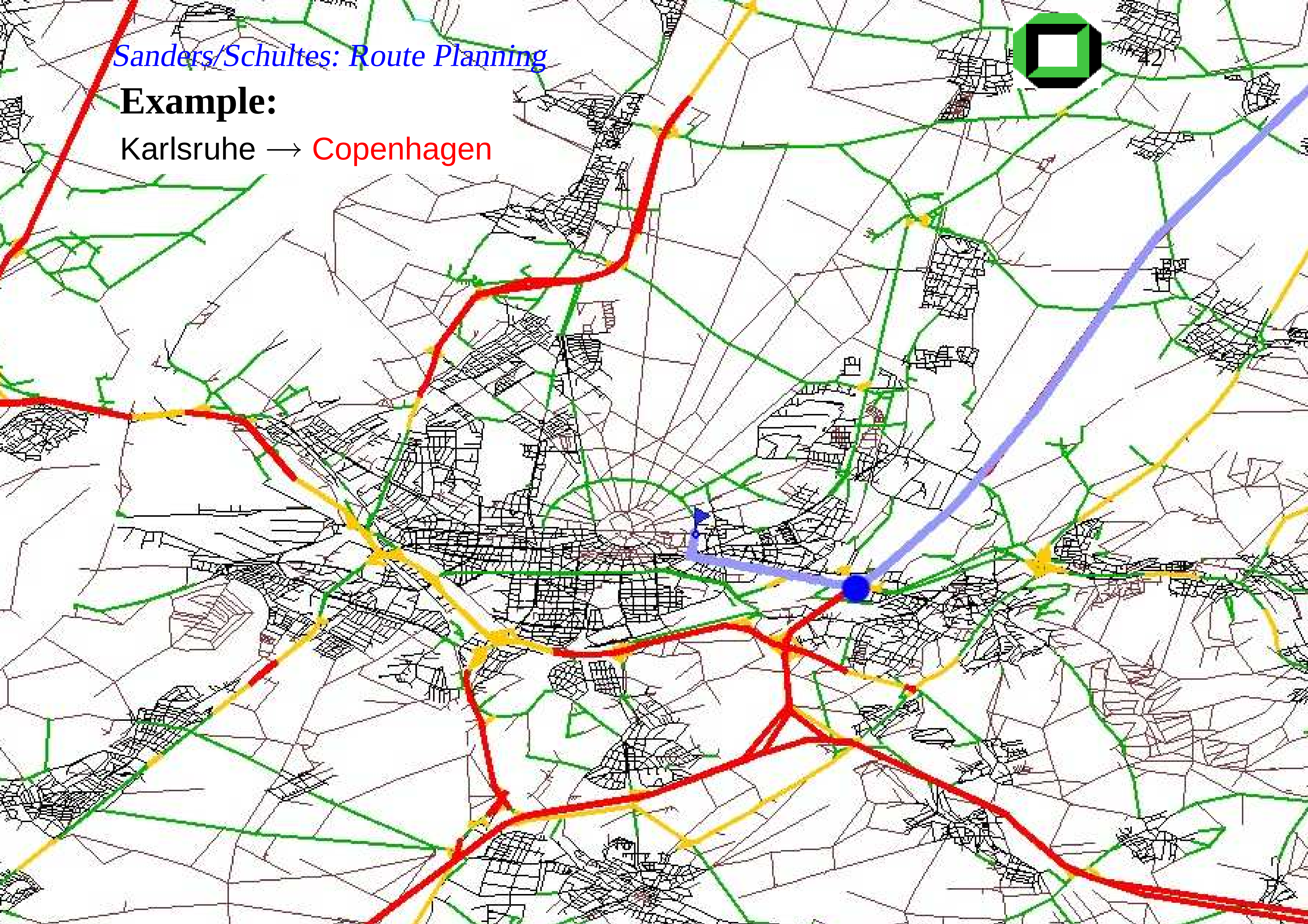
Example:

Karlsruhe $\longrightarrow$ Brussels



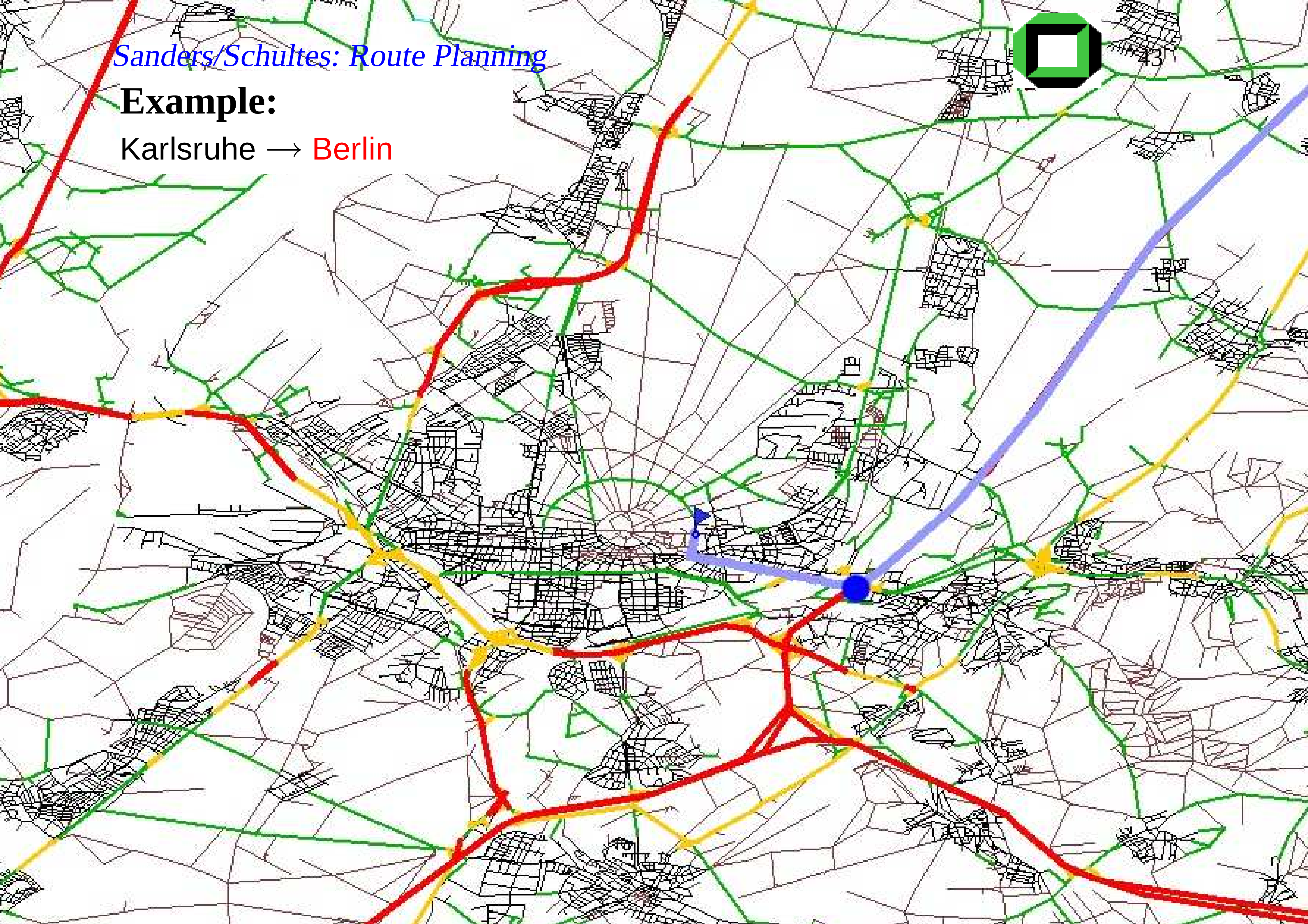
Example:

Karlsruhe → Copenhagen



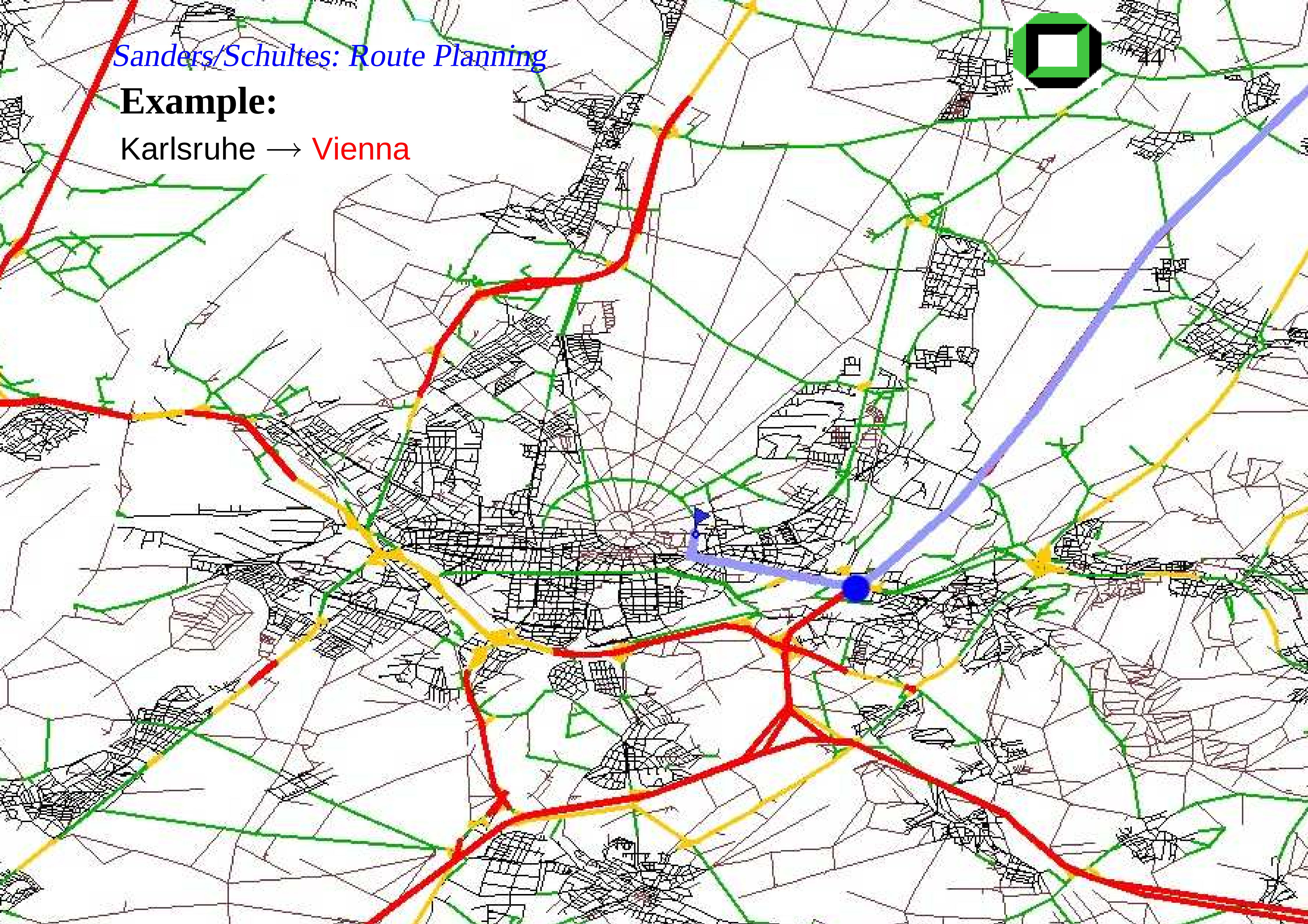
Example:

Karlsruhe → Berlin



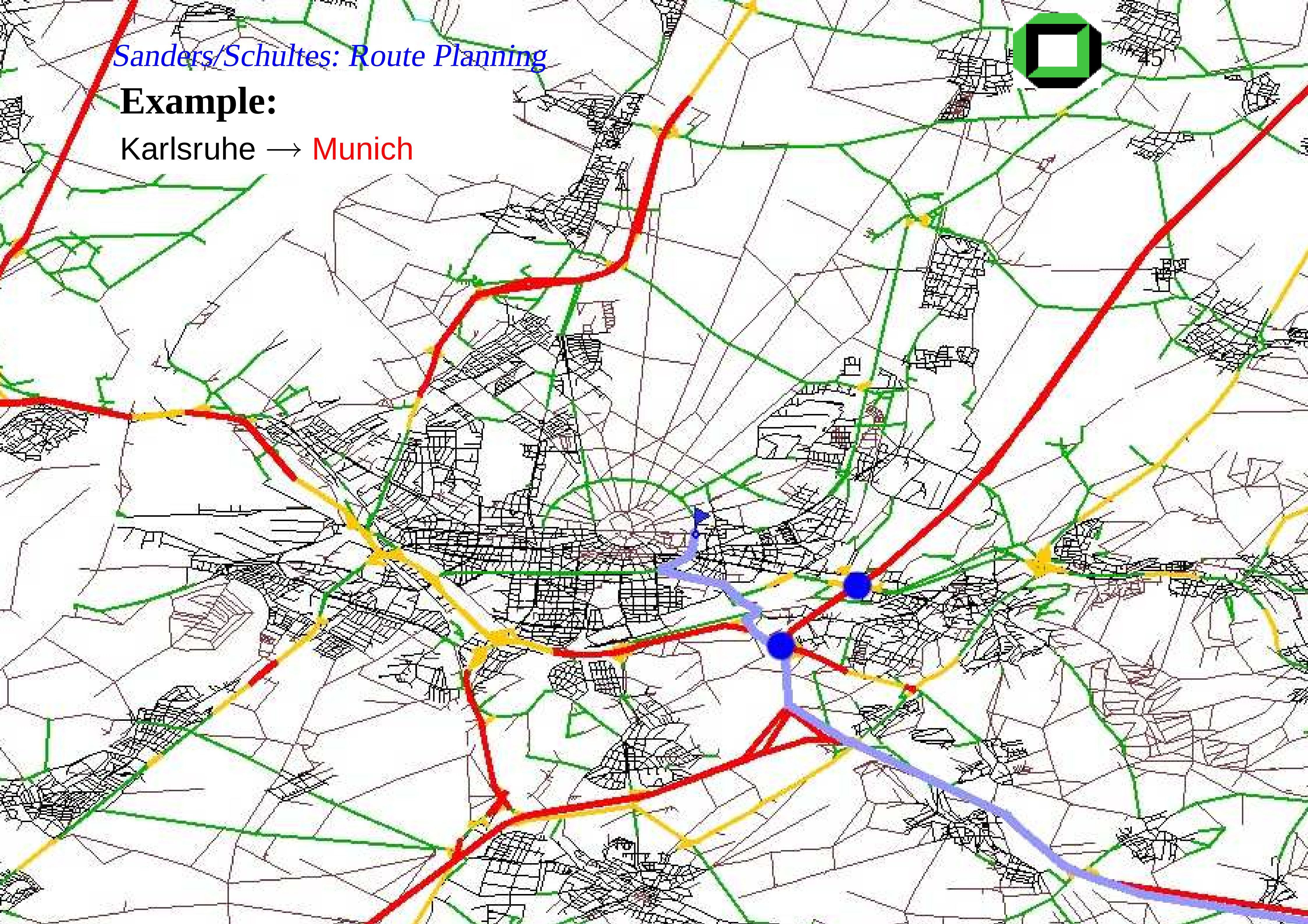
Example:

Karlsruhe → Vienna



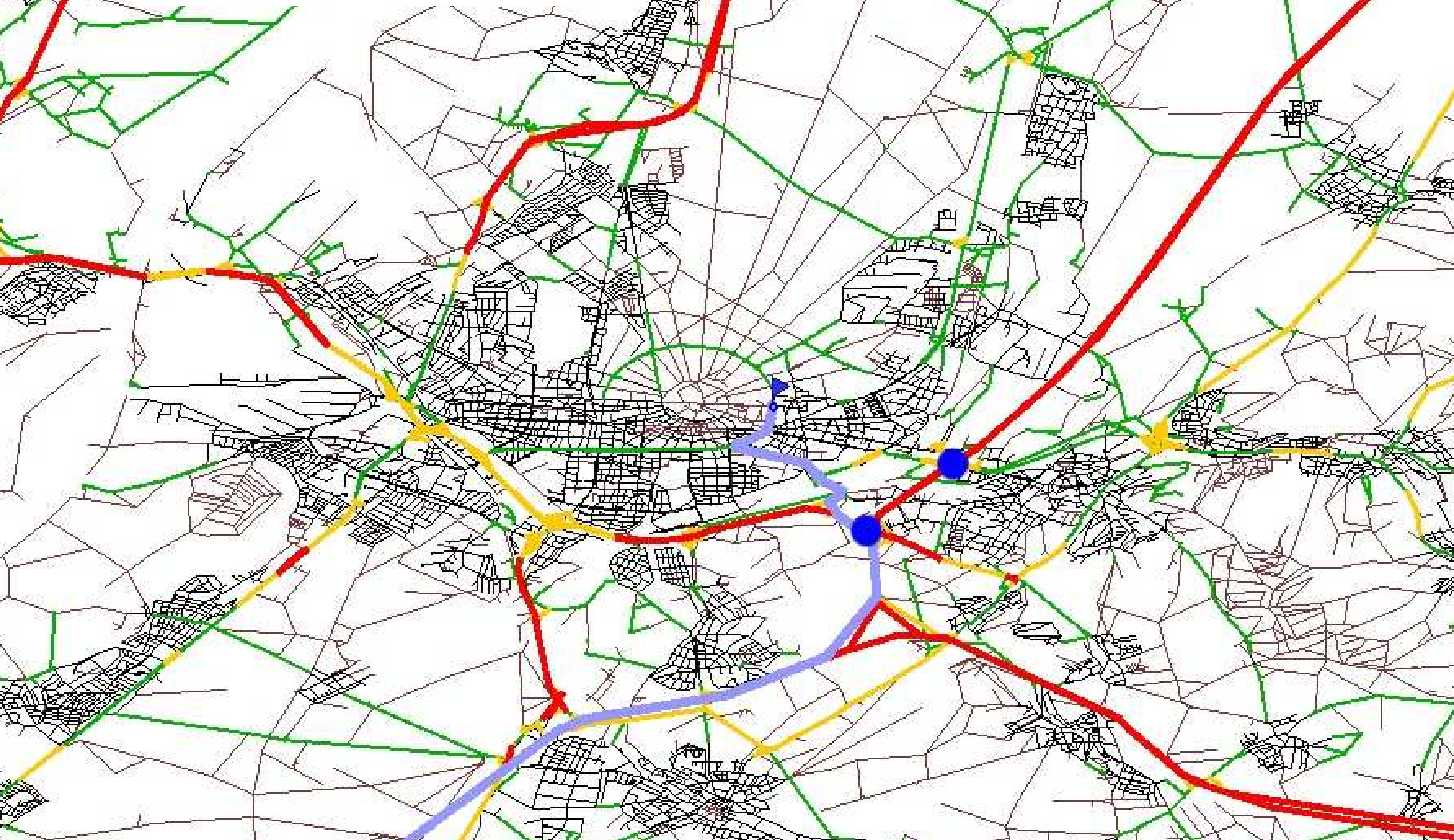
Example:

Karlsruhe → Munich



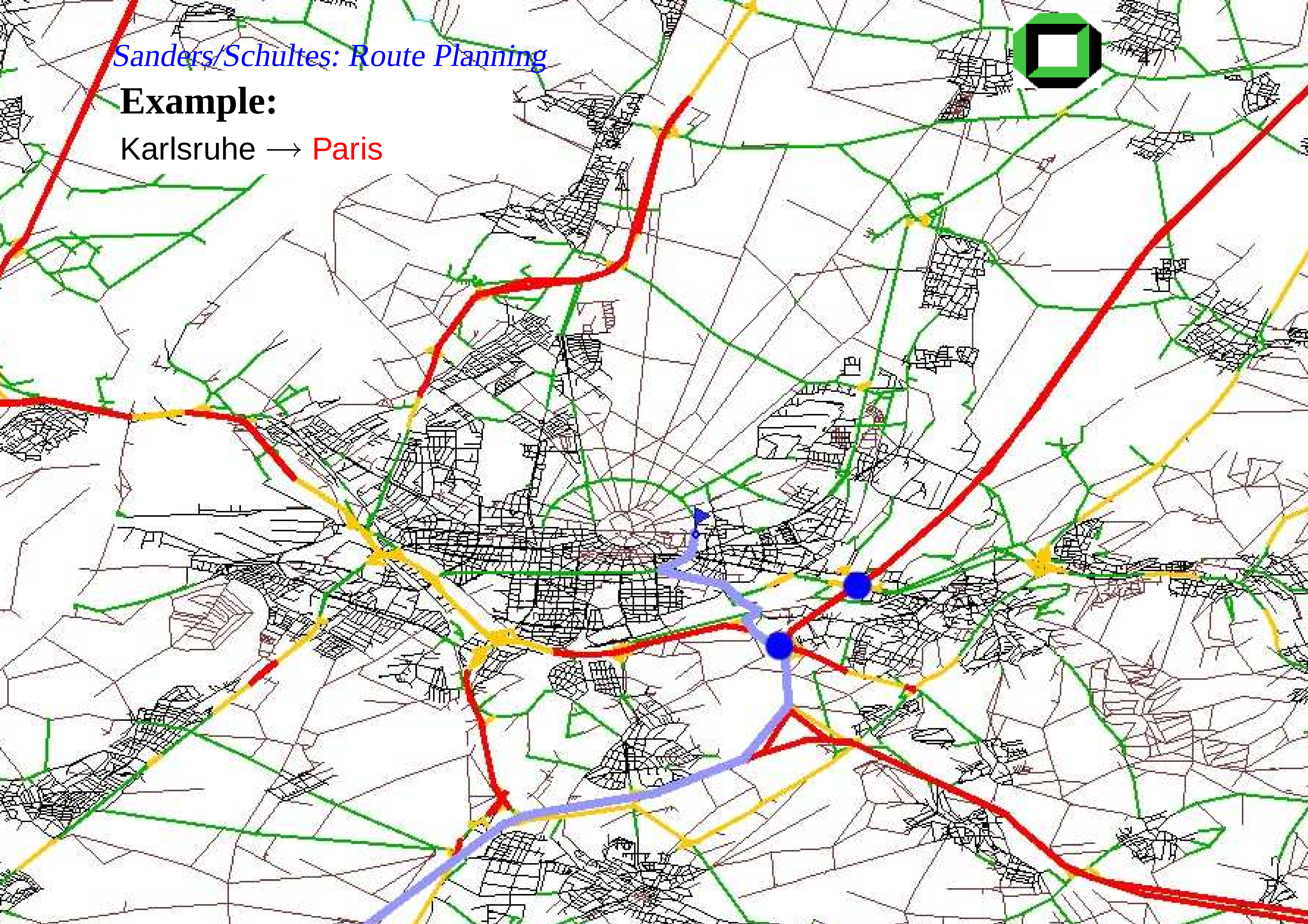
Example:

Karlsruhe → Rome



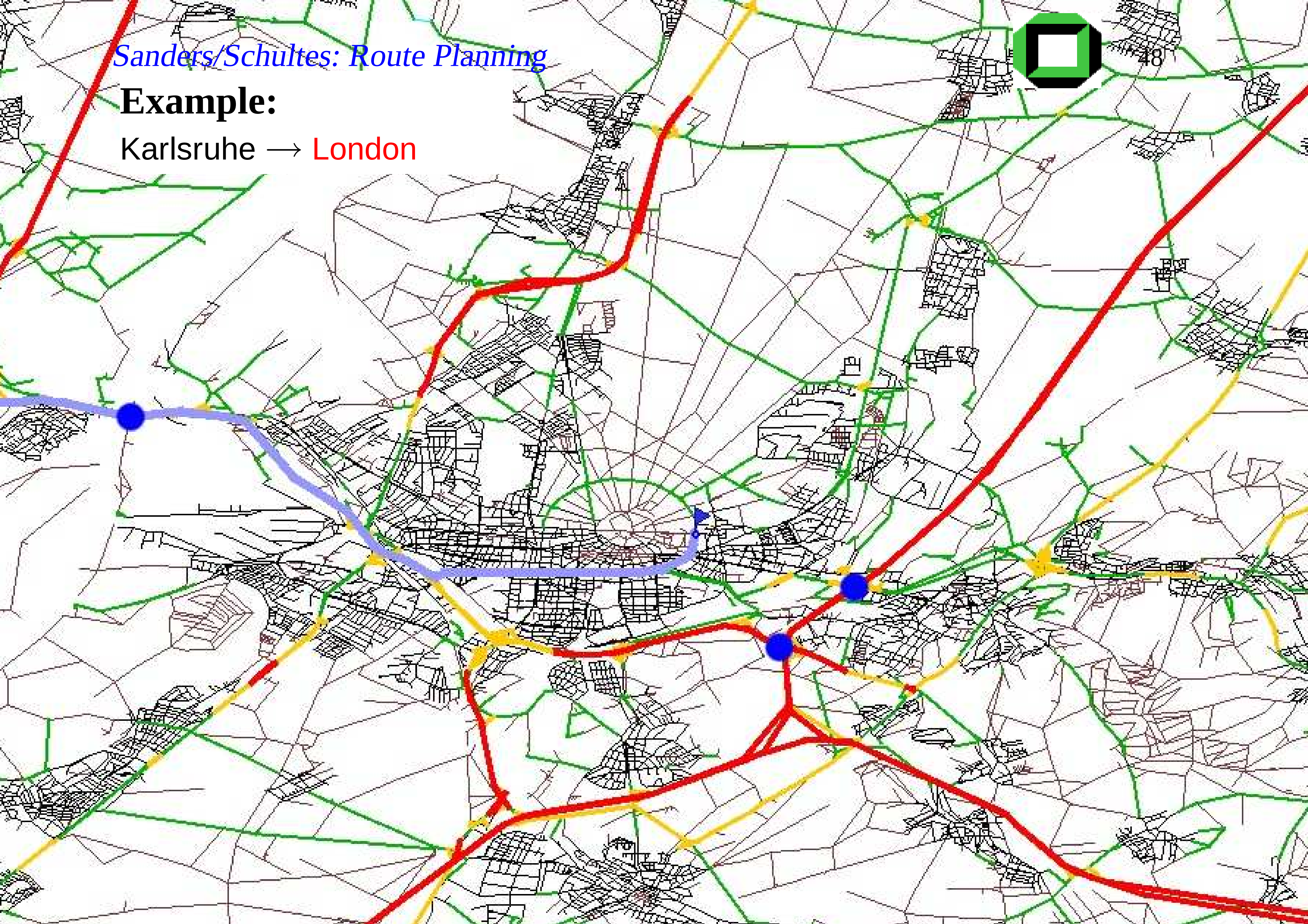
Example:

Karlsruhe → Paris



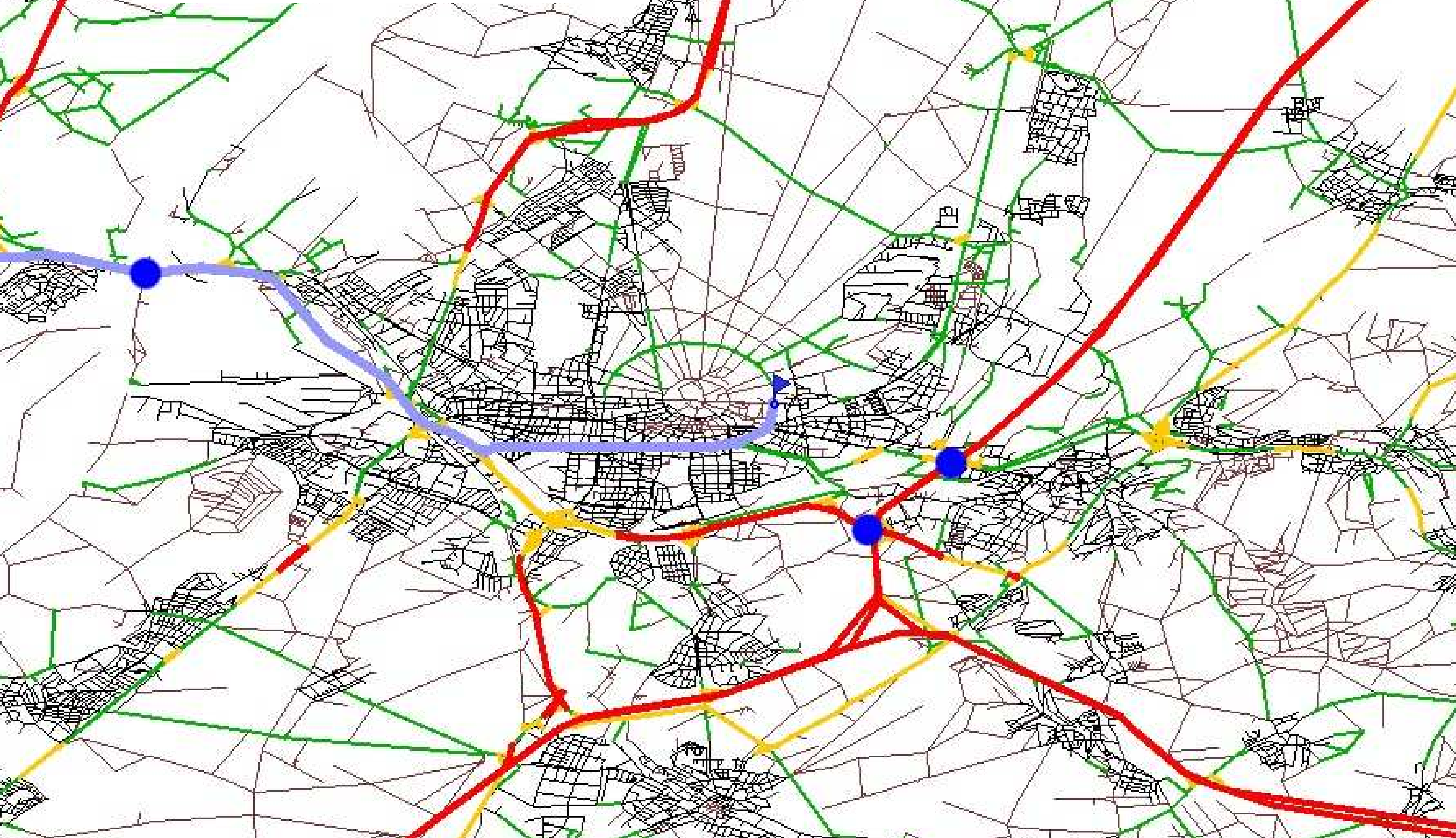
Example:

Karlsruhe → London



Example:

Karlsruhe ⟶ Brussels



Observations for long-distance travel

Europe

1. leave area via one of only a few ‘important’ traffic junctions,

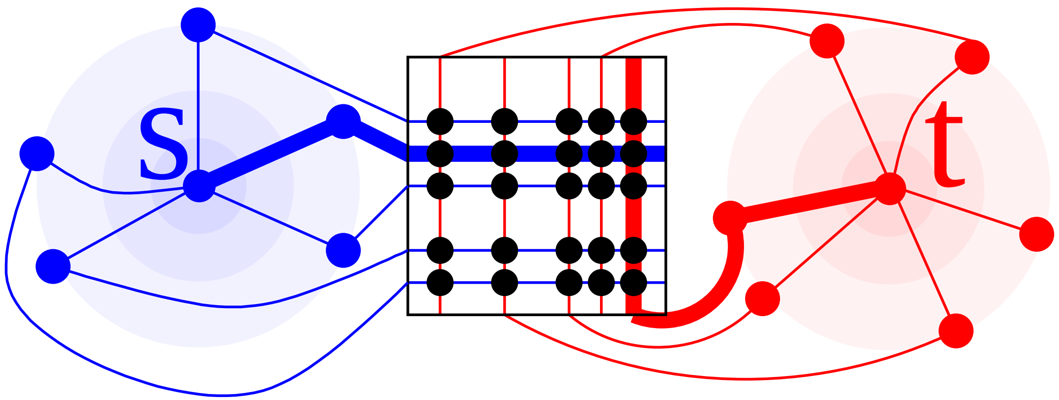
   called **access points**. ≈10

   ⇝ we can store all access points for each node

2. union of the access points of all nodes is small,

   called **transit-node set**. ≈10 000

   ⇝ we can store the distances between all transit-node pairs



Transit-Node Routing

Preprocessing:

- identify transit-node set $\mathcal{T} \subseteq V$
- compute complete $|\mathcal{T}| \times |\mathcal{T}|$ distance table
- for each node: identify its access points (mapping $A : V \longrightarrow 2^{\mathcal{T}}$),
  store the distances

Query (source s and target t given): compute

$$d_{\text{top}}(s,t) := \min\{d(s,u) + d(u,v) + d(v,t) : u \in A(s), v \in A(t)\}$$

Transit-Node Routing

Locality Filter:

local cases must be filtered ($\leadsto$ special treatment)

$L : V \times V \rightarrow \{\text{true}, \text{false}\}$

$$\neg L(s,t) \text{ implies } d(s,t) = d_{\text{top}}(s,t)$$

Additional Layers:

Local cases: use secondary transit-node set.

secondary distance table:

store only distances between “nearby” secondary transit-nodes.

. . . secondary locality filter, tertiary transit-nodes,. . .

Base case: very limited local search

Our Implementation

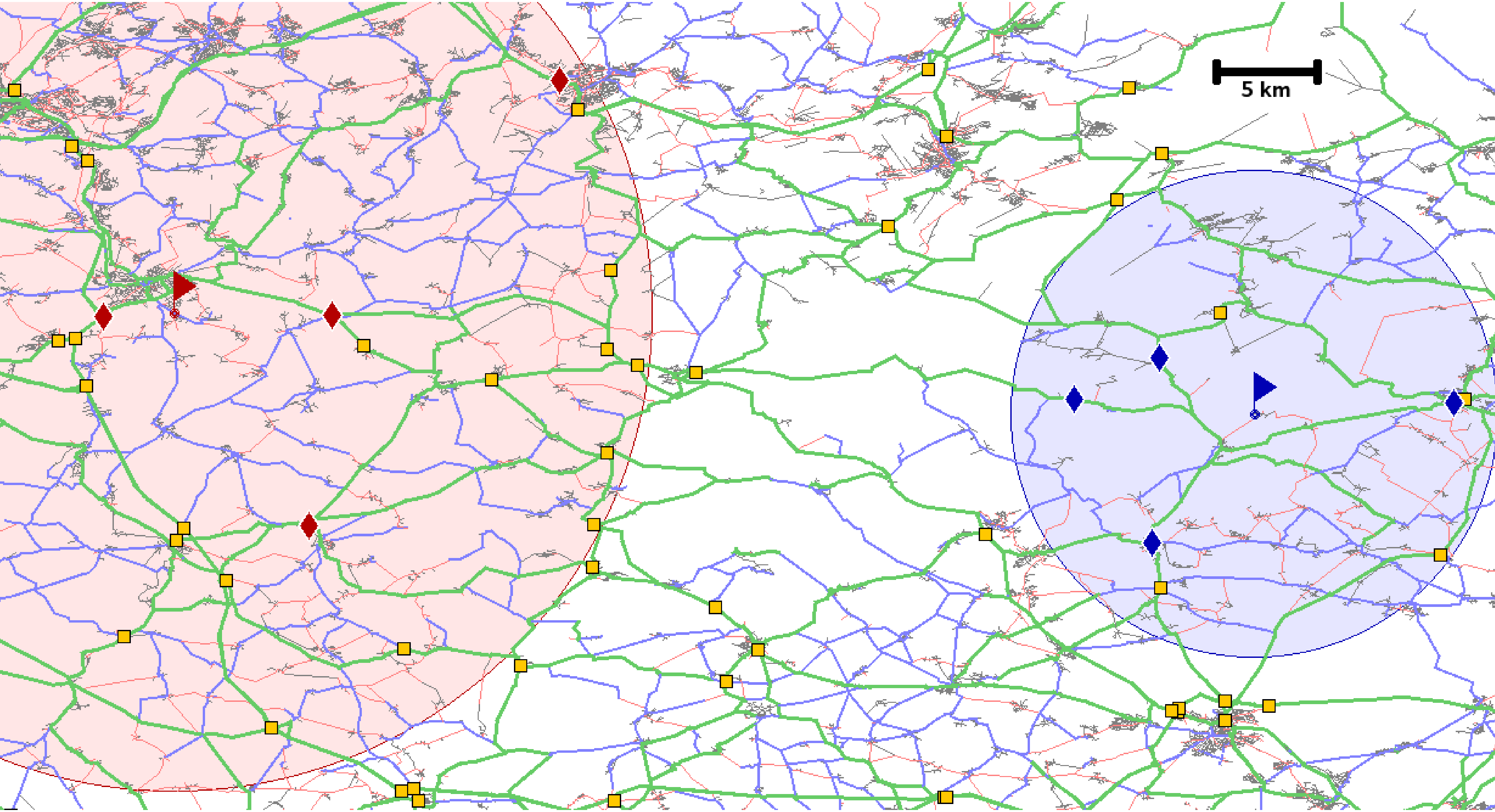
transit-node sets: appropriate levels of highway hierarchy (1–3 layers)

access nodes: minimization step, e.g., $\approx 55 \longrightarrow \approx 10$

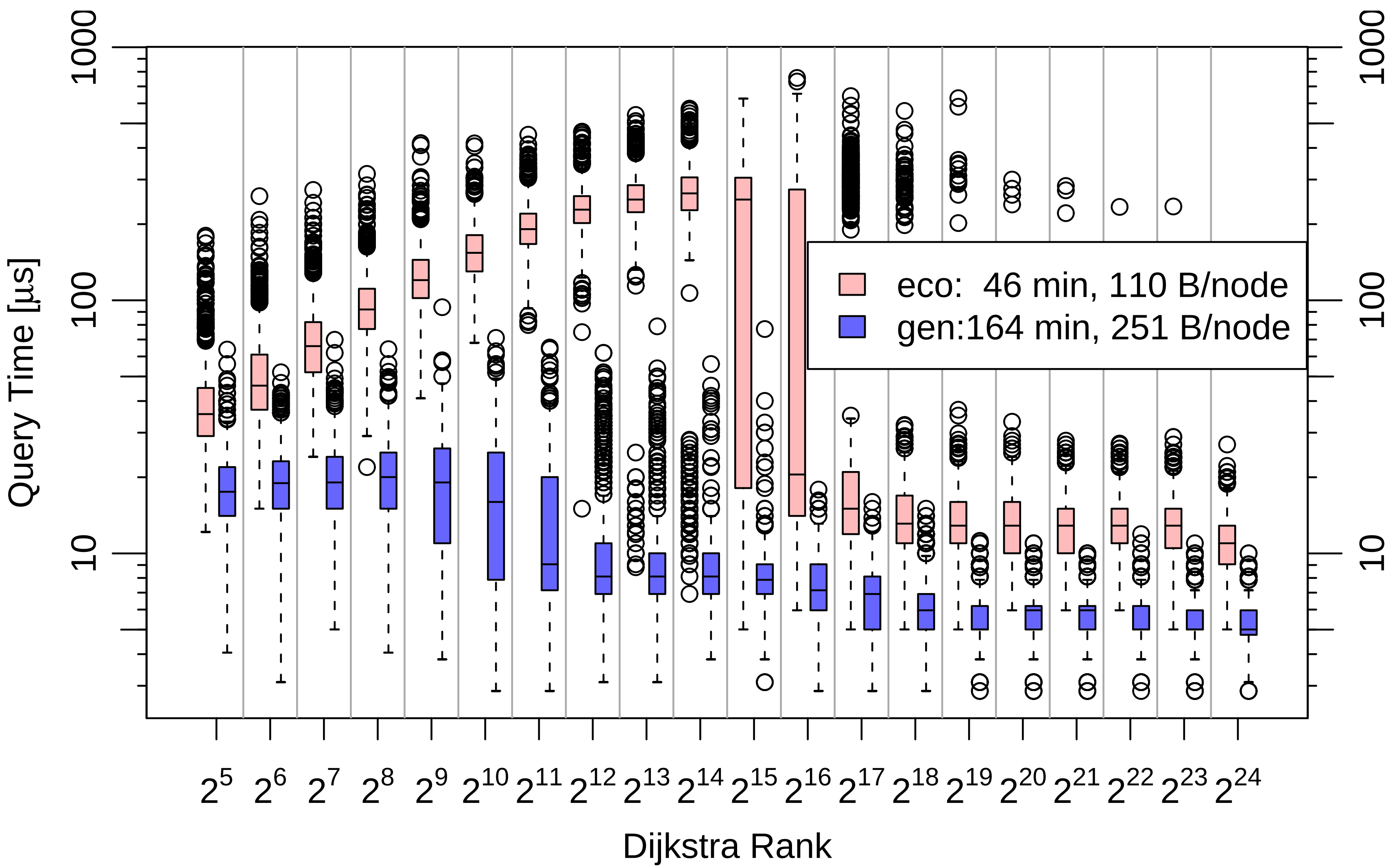
locality filter: geometric disks around s and t intersect ?

distance tables: (generalized) many-to-many routing

Example



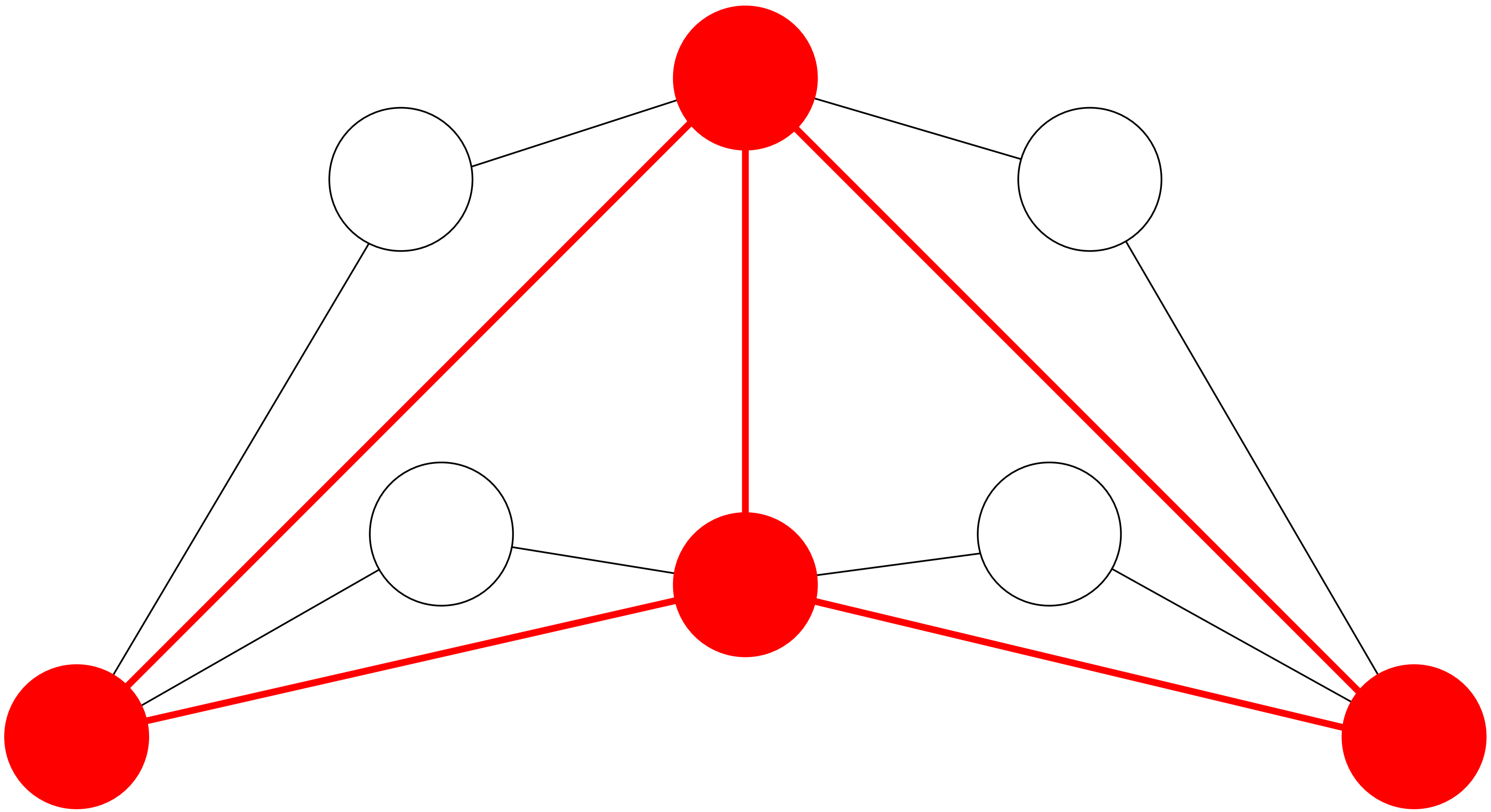
Local Queries (Transit-Node Routing, Europe)



3. Approach

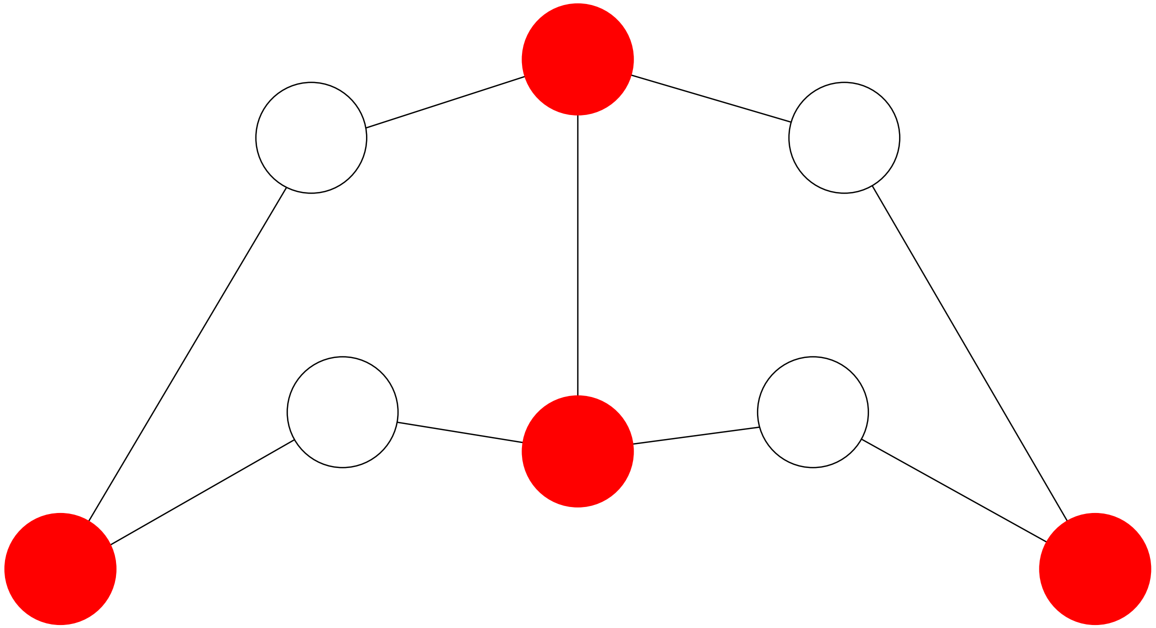
Highway-Node Routing

[SS 07–]



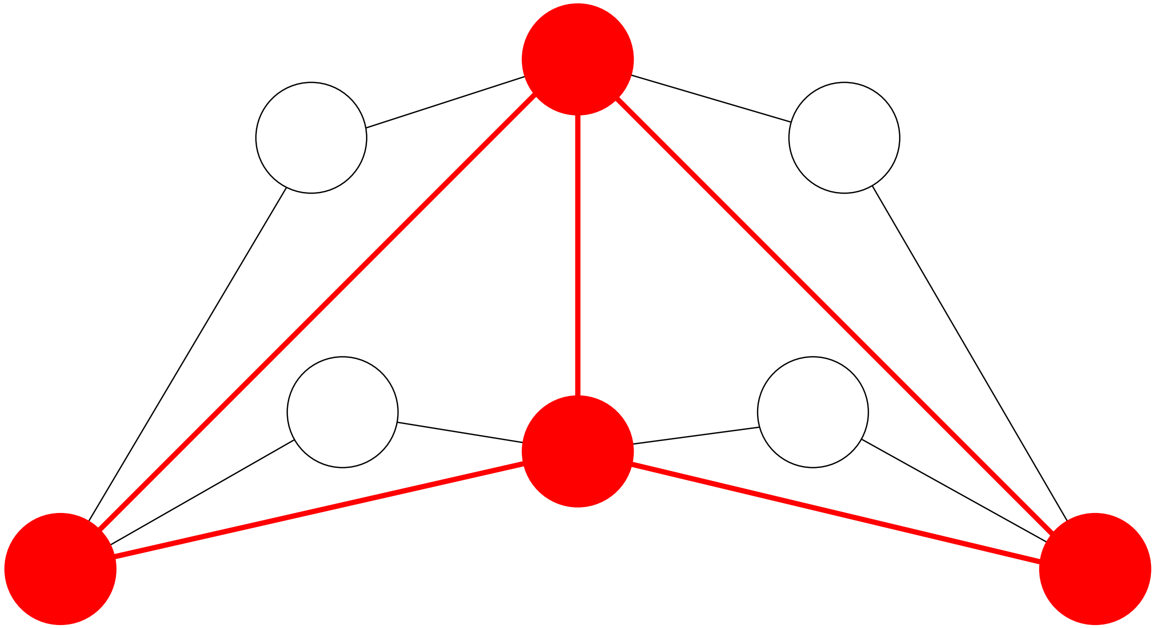
Highway-Node Routing

- classify nodes according to 'importance' (use hwy hierarchies)



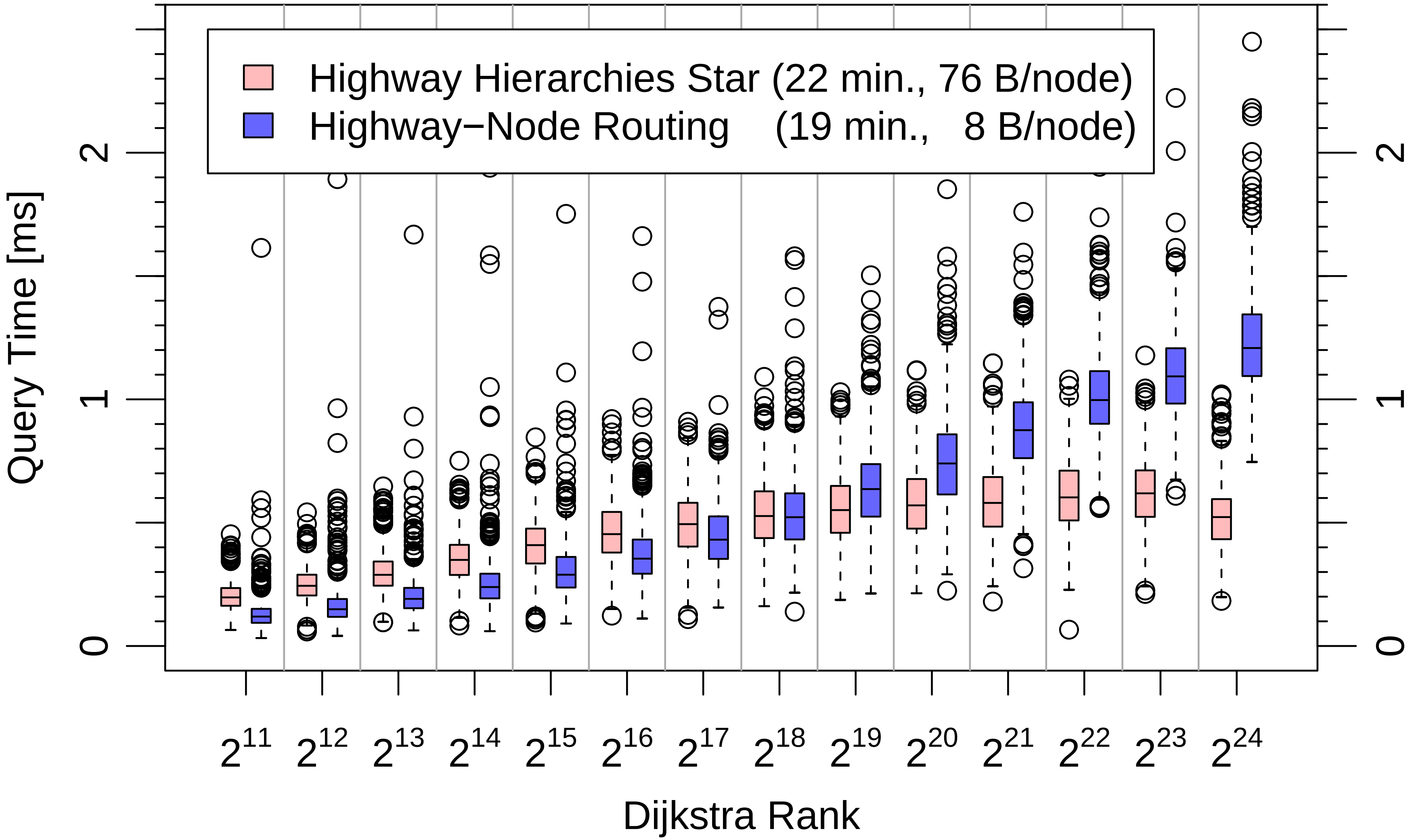
Highway-Node Routing

- classify nodes according to ‘importance’ (use hwy hierarchies)



- perform queries in (multi-level) overlay graphs

Static Highway-Node Routing (Europe)



Dynamic Highway-Node Routing

- change entire cost function

  typically < 2 minutes

- change a few edge weights
  - update data structures

    2 – 40 ms per changed edge

  OR

  - perform prudent query

    e.g., 47.5 ms if 100 motorway edges have been changed

Summary

Highway Hierarchies: Fast routing, fast preprocessing, low space, few tuning parameters, basis for many-to-many, transit-node routing, highway-node routing.

Many-to-Many: Huge distance tables are tractable. Subroutine for transit-node routing.

Transit-Node Routing: Fastest routing so far.

Highway-Node Routing: “Simpler” HHs, fast routing, very low space, efficiently dynamizable.

Future Work I: More on Static Routing

- Better choices for transit-node sets or highway-node sets.
  (use centrality measures, separators, explicit optimization,. . .)
- A hierarchical routing scheme that allows stopping bidirectional search earlier ? (competetive with HHs, HNR)
- Better integration with goal directed methods.
  (PCDs, A^*, edge flags, geometric containers)
- Experiments with other networks.
  (communication networks, VLSI, social networks, computer games, geometric problems, . . .)
- Specialized preprocessing for one batch of (many-to-many) queries

Future Work II: Theory Revisited

- Correctness proofs
- Stronger impossibility results (worst case)
- Analyze speedup techniques for model graphs
- Characterize graphs for which a particular (new?) speedup technique works well
- A method with low worst-case query time, but preprocessing might become quadratic ?

Future Work III: Towards Applications

- Turn penalties (implicitly represented)

  Just bigger but more sparse graphs ?

- Parallelization (server scenarios, logistics, traffic simulation)

  easy (construction, many-to-many, many queries)

- Mobile platforms

  $\rightsquigarrow$ adapt to memory hierarchy (RAM $\leftrightarrow$ flash)

  $\rightsquigarrow$ data compression

Future Work IV: Beyond Static Routing

- More dynamic routing (e.g. for transit-node routing)
- Time-dependent networks

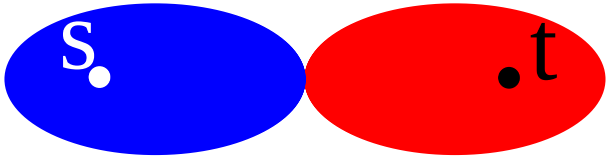
  (public transportation, traffic-dependent travel time)
- Preprocessing for an entire spectrum of objective functions
- Multi-criteria optimization

  (time, distance, fuel, toll, driver preferences,. . .)
- Approximate traffic flows

  (Nash-equilibria, (fair) social optima)
- Traffic steering (road pricing, . . .)

Appendix

Goal-Directed Search



A^* [Hart, Nilsson, Raphael 68]: not effective for travel time

Geometric Containers [Wagner et al. 99–05]:

high speedup but quadratic preprocessing time

Landmark A^* [Goldberg et al. 05–]: precompute distances to $\approx$ 20

landmarks $\rightsquigarrow$ moderate speedups, preprocessing time, space

Precomputed Cluster Distances [S, Maue 06]:

more space-efficient alternative to landmarks

Hierarchical Methods

Planar graph (theory) [Fakcharoenphol, Rao, Klein 01–06]: $O(n \log^2 n)$

space and preprocessing time; $O(\sqrt{n} \log n)$ query time

Planar approximate (theory) [Thorup 01]: $O((n \log n)/\varepsilon)$ space and

preprocessing time; almost constant query time

Separator-based multilevel [Wagner et al. 99–]:

works, but does not capitalize on importance induced hierarchy

Reach based routing [Gutman 04]:

elegant, but initially not so successful

Highway hierarchies [SS 05–]: stay tuned

Advanced reach [Goldberg et al. 06–]: combinable with landmark A^*

Transit-node routing [Bast, Funke, Matijevic, S, S 06–]: stay tuned

Highway-node routing [SS 07–]: stay tuned

An Algorithm Engineering Perspective

Models: Preprocessing, point-to-point, dynamic, many-to-many
parallel, memory hierarchy, time dependent, multi-objective,...

Design: HHs, HNR, transit nodes,... wide open

Analysis: Correctness, per instance. big gap

Implementation: tuned, modular, thorough checking, visualization.

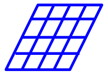
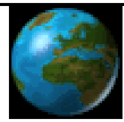
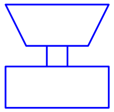
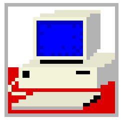
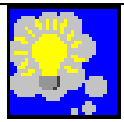
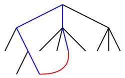
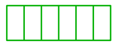
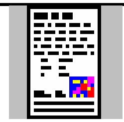
Experiments: Dijkstra ranks, worst case, cross method....

Instances: Large real world road networks.
turn penalties, queries, updates, other network types

Algorithm Libraries: ???

Applications: Promising contacts, hiring. more should come.

Gaps Between Theory & Practice

Theory	$\longleftrightarrow$	**Practice**
simple 	**appl. model**	 complex
simple 	**machine model**	 complex
complex 	**algorithms**	FOR simple
complex 	**data structures**	 simple
worst case max	**complexity measure**	 inputs
asympt. $O(\cdot)$	**efficiency**	42% constant factors

Goals

- bridge gaps between theory and practice
- accelerate transfer of algorithmic results into applications
- keep the advantages of theoretical treatment:

  generality of solutions and

  reliabiltiy, predictabilty from performance guarantees

Canonical Shortest Paths

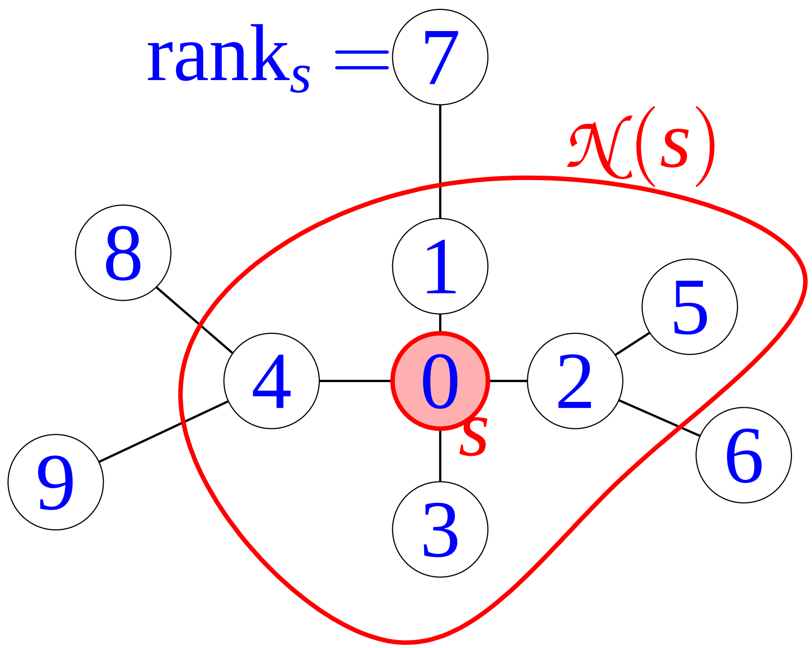
$\mathcal{SP}$: Set of shortest paths

$\mathcal{SP}$ canonical $\Leftrightarrow$

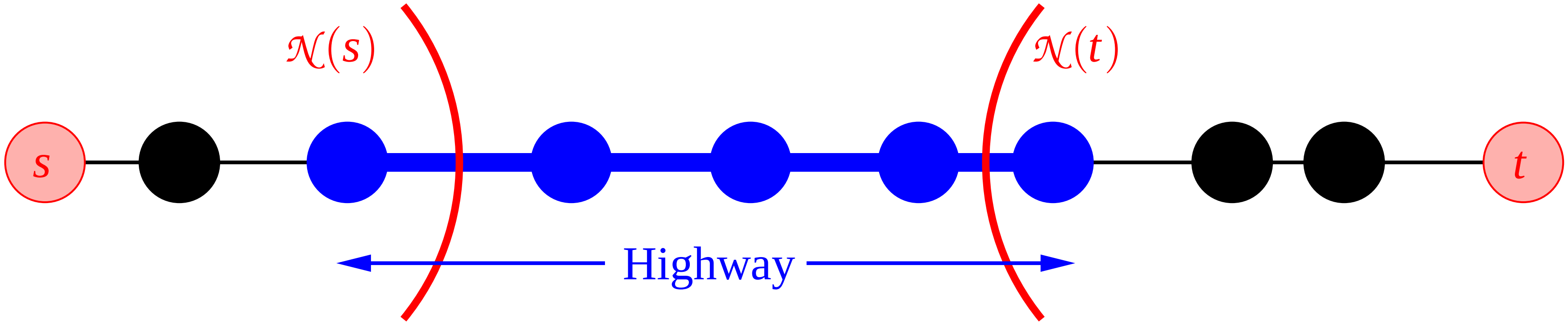
$$\forall P = \langle s, \ldots, s', \ldots, t', \ldots, t \rangle \in \mathcal{SP} : \langle s' \rightarrow t' \rangle \in \mathcal{SP}$$

A Meaning of "Local"

- choose neighbourhood radius $r(s)$
  e.g. distance to the H-closest node for a fixed parameter H
- define neighbourhood of s:
  $\mathcal{N}(s) := \{v \in V \mid d(s,v) \le r(s)\}$
- example for $H = 5$



Highway Network



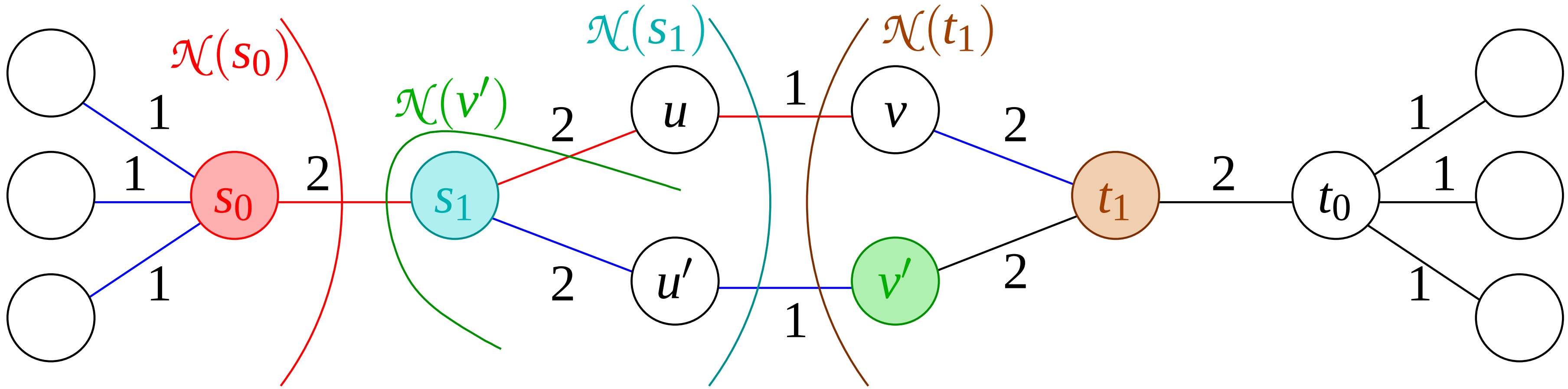
Edge (u,v) belongs to highway network *iff* there are nodes s and t s.t.

- (u,v) is on the "*canonical*" shortest path from s to t

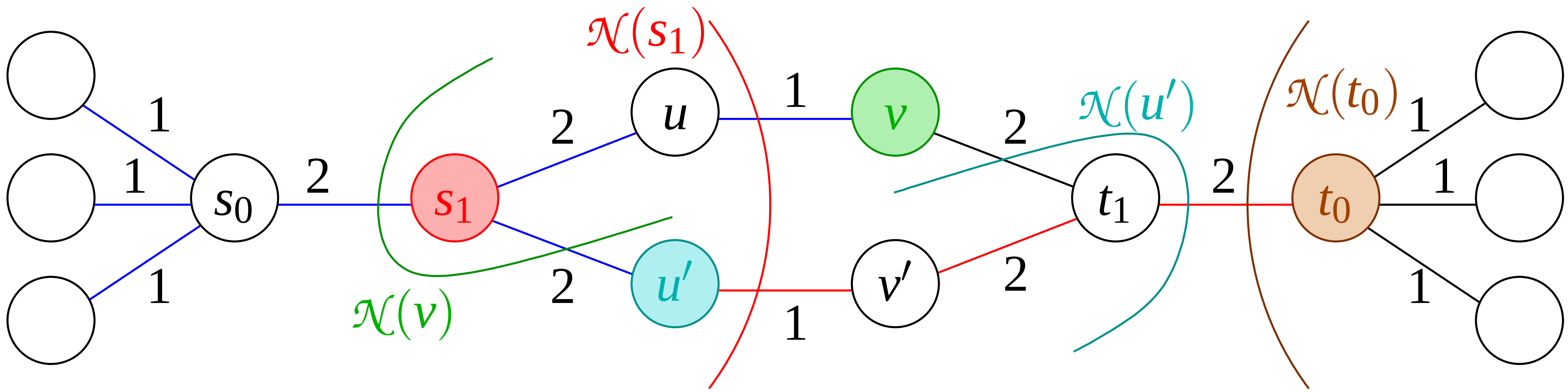
and

- (u,v) is not entirely within $\mathcal{N}(s)$ or $\mathcal{N}(t)$

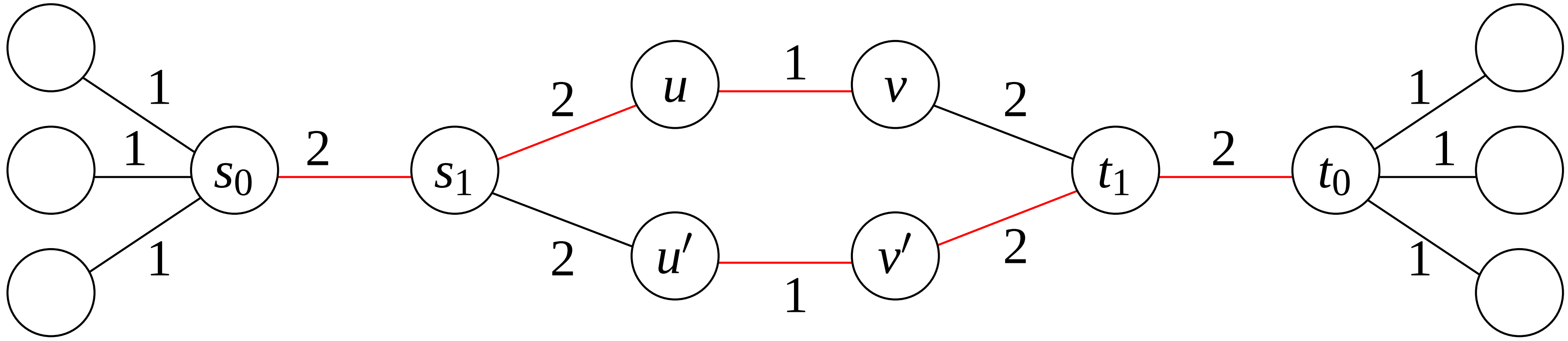
Canonical Shortest Paths



(a) Construction, started from s_0.

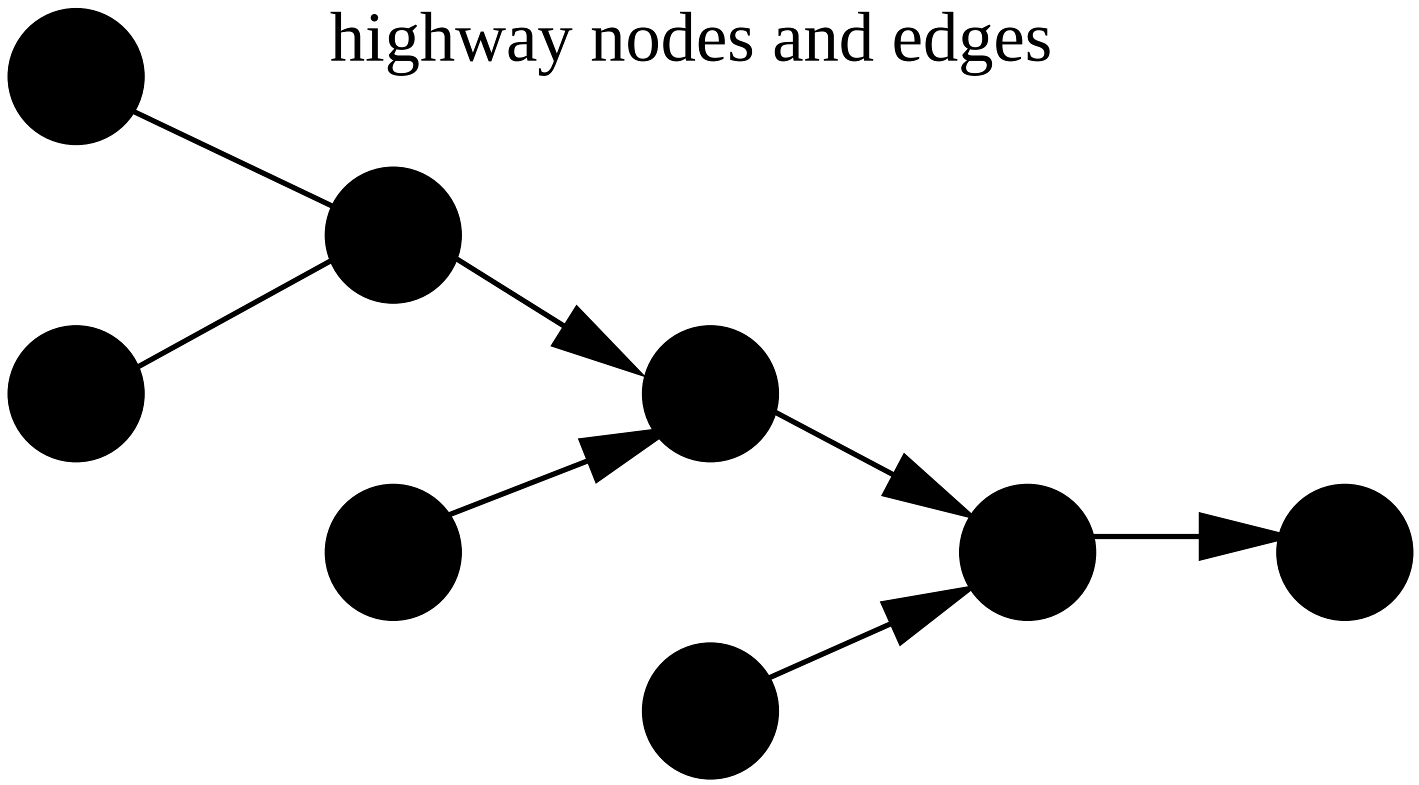


(b) Construction, started from s_1.

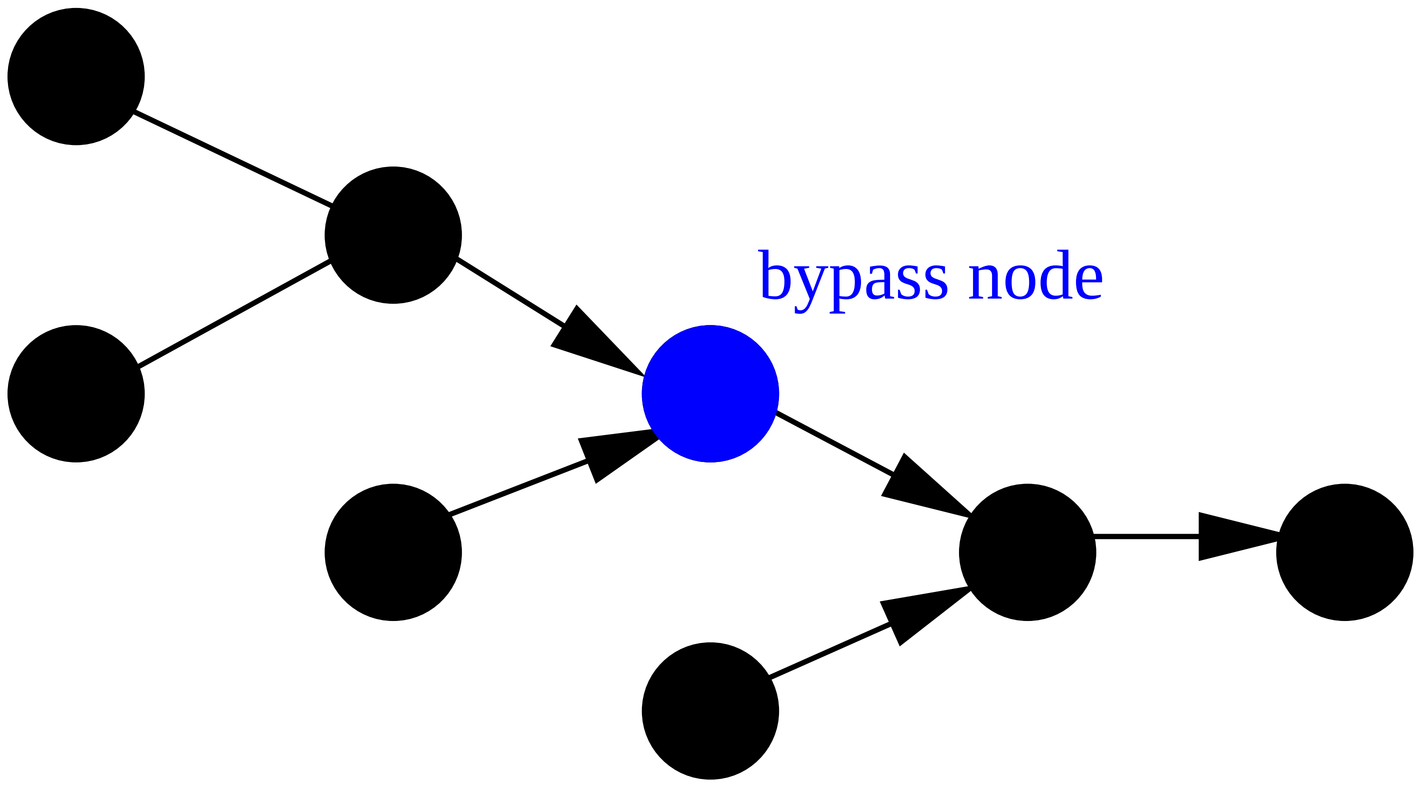


(c) Result of the construction.

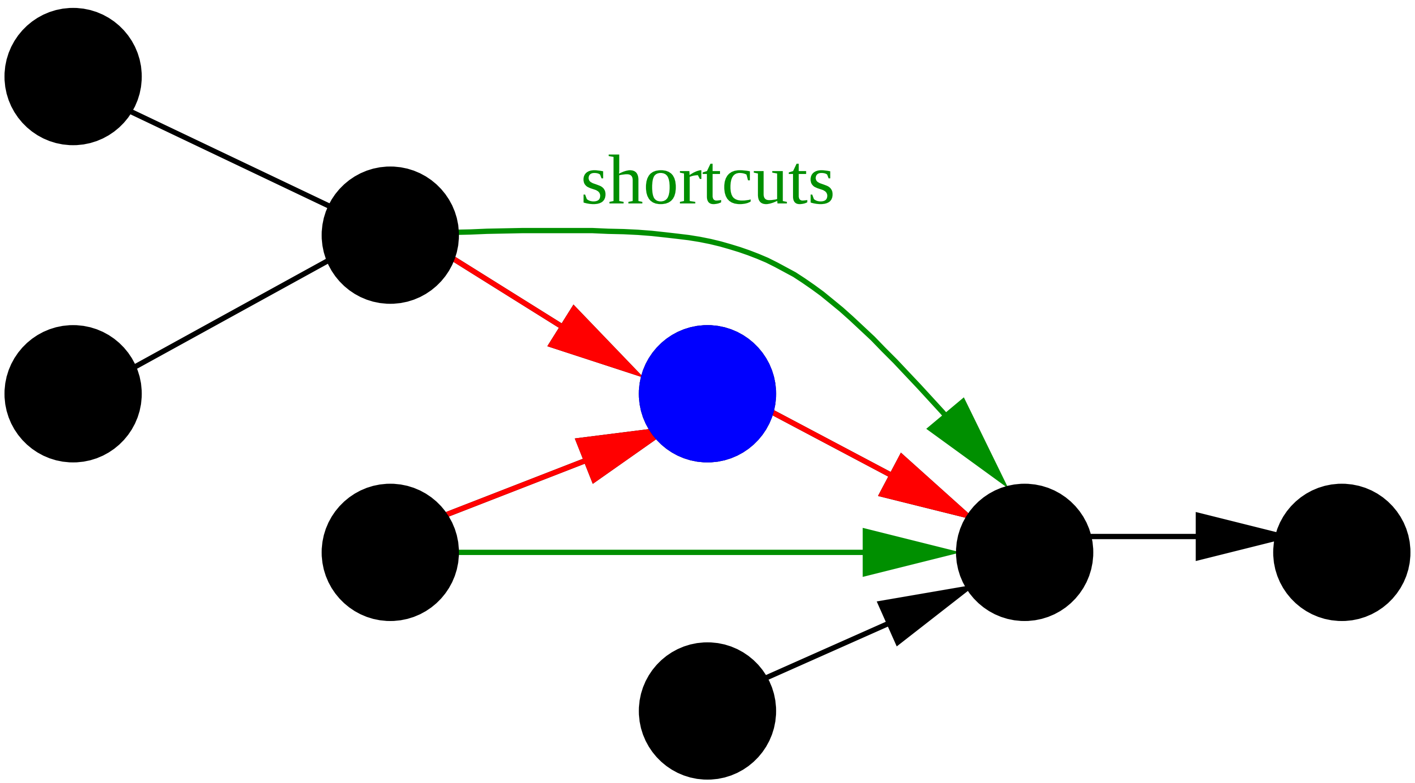
Contraction



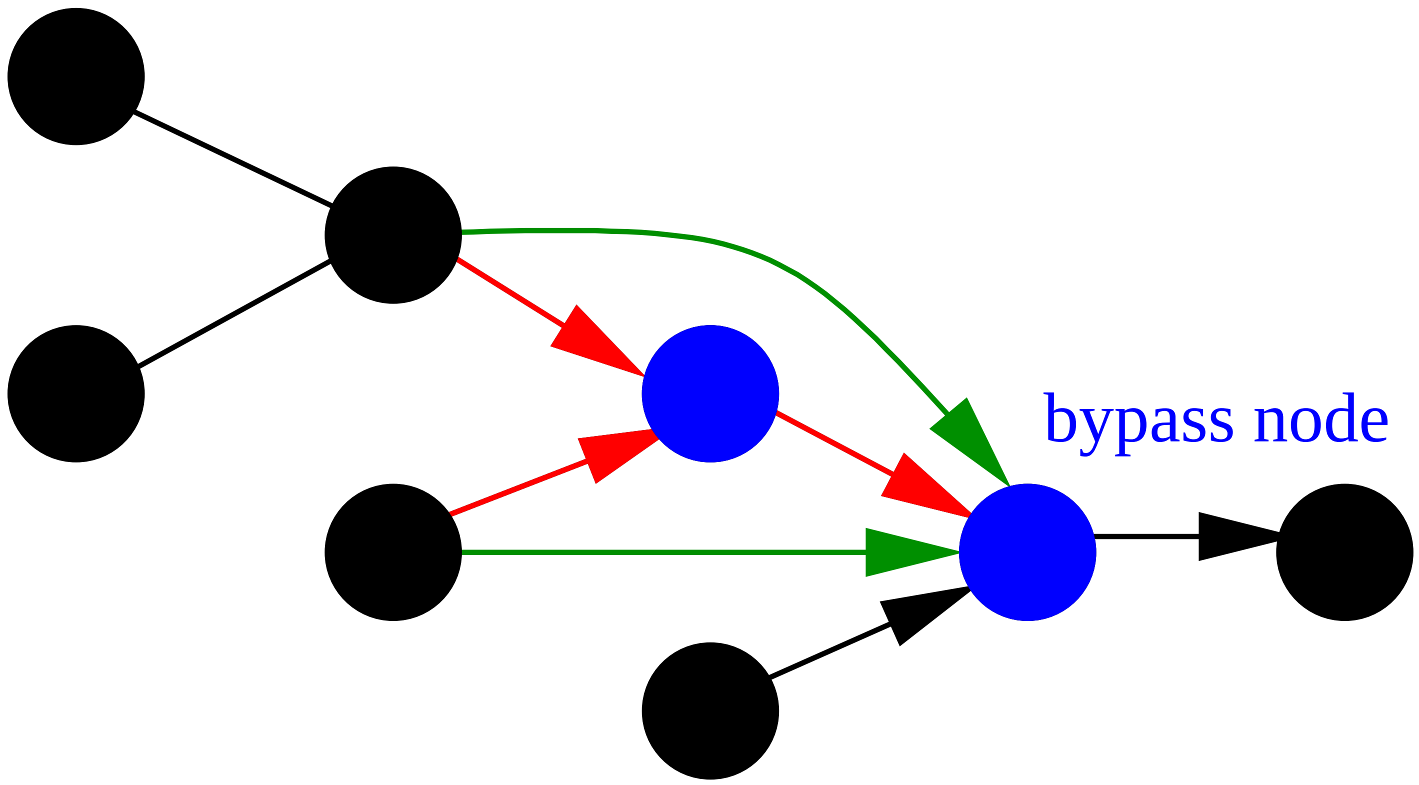
Contraction



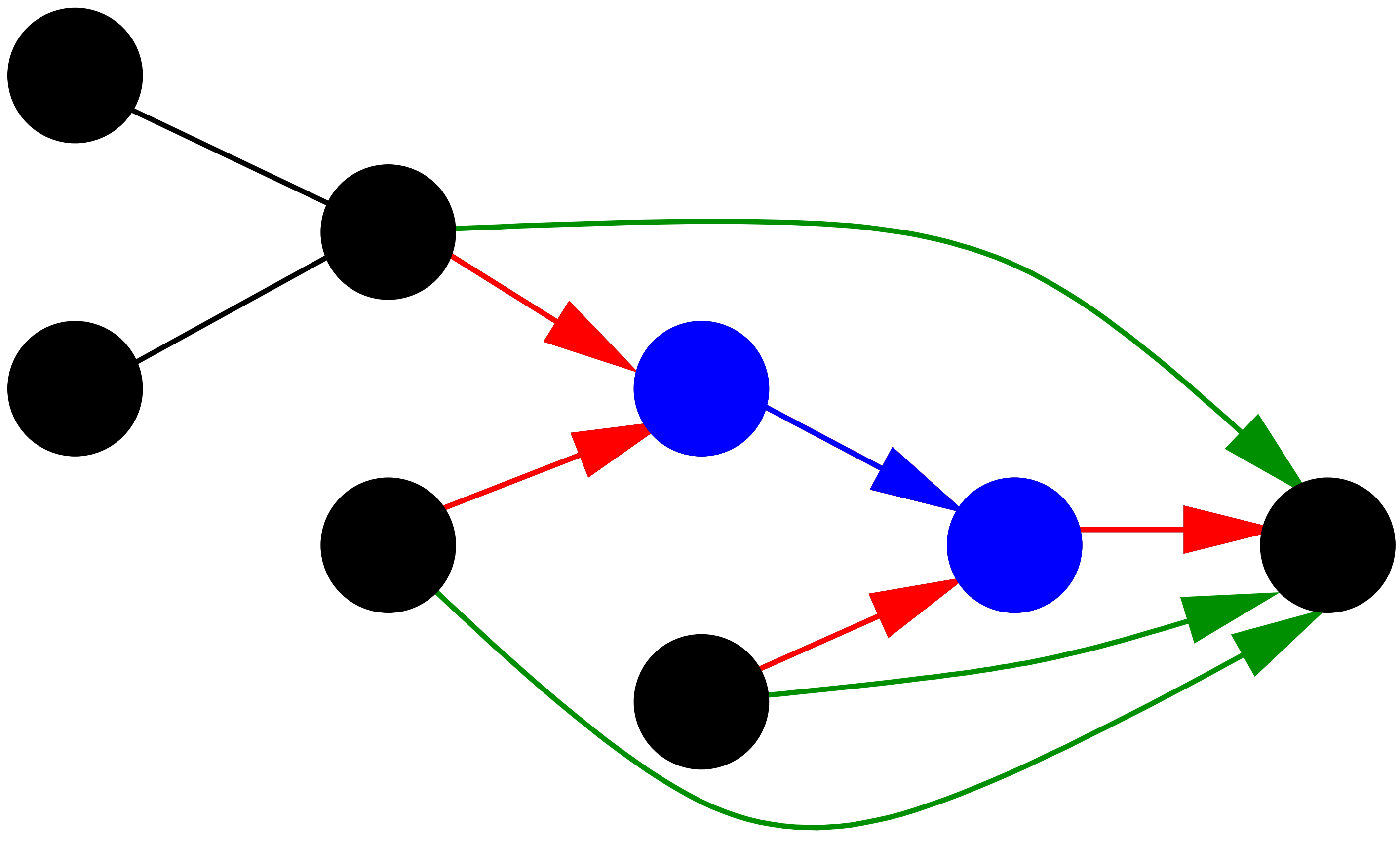
Contraction



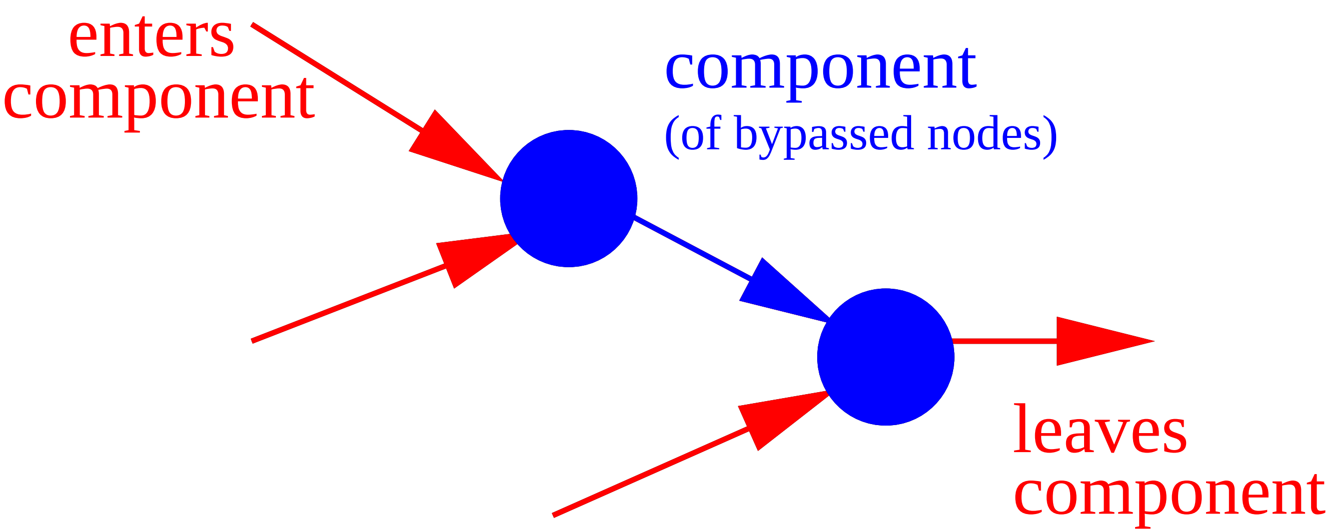
Contraction



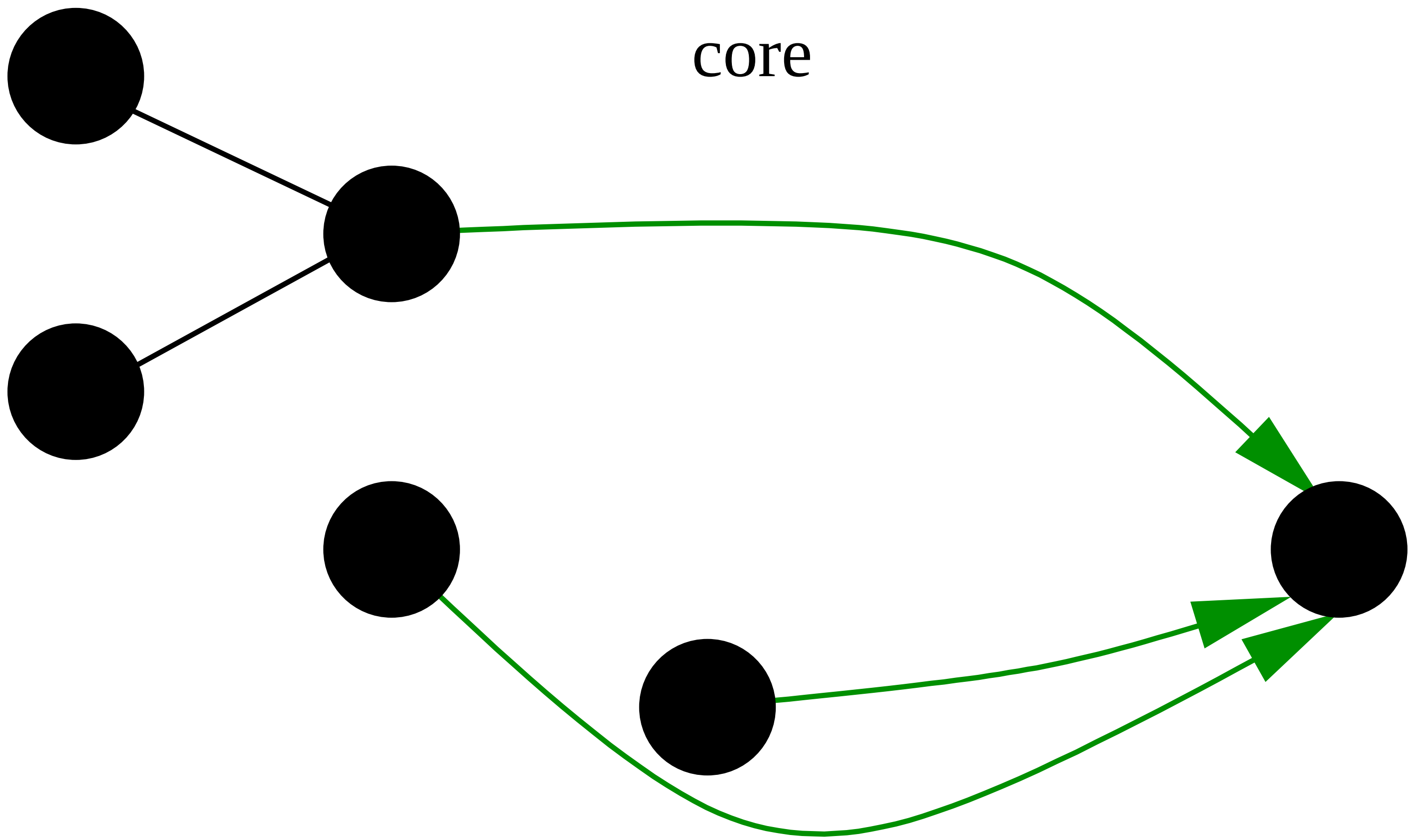
Contraction



Contraction



Contraction



Contraction

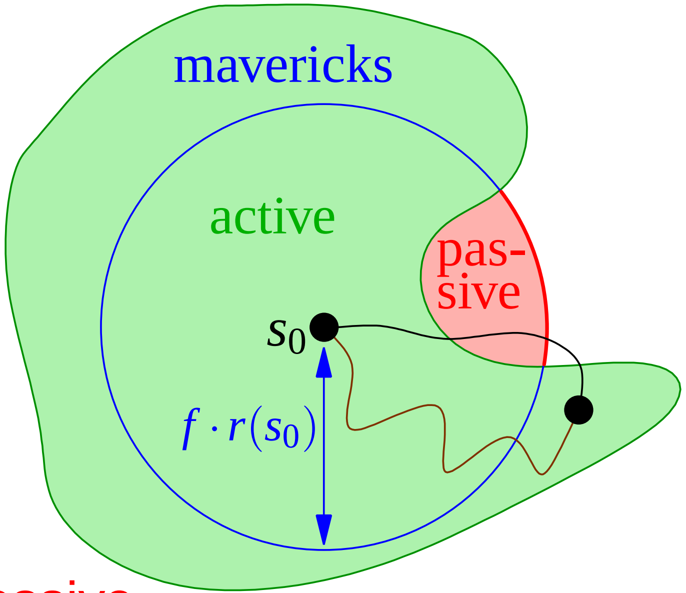
Which nodes should be bypassed?

Use some heuristic taking into account

- the number of shortcuts that would be created and
- the degree of the node.

Fast Construction of the Highway Network

Look for HH-edges only in (modified) local SSSP search trees.



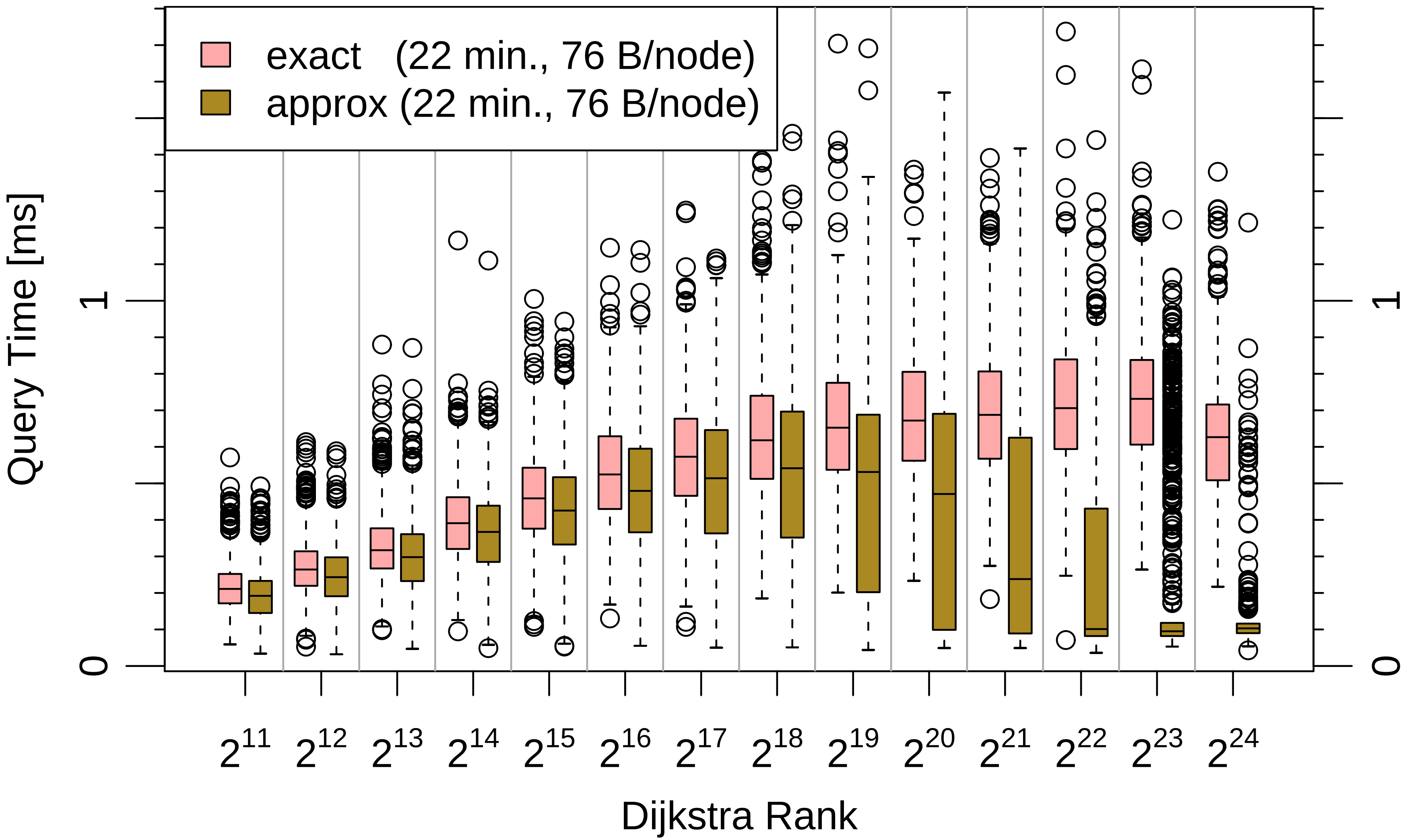
- Nodes have state

  active, passive, or mavericks.
- s_0 is active.
- Node states are inherited

  from parents in the SSSP tree.
- abort condition$(p) \longrightarrow p$ becomes passive.
- $d(s_0, p) > f \cdot r(s_0) \longrightarrow p$ becomes maverick.
- all nodes maverick? $\longrightarrow$ stop searching from passive nodes
- all nodes passive or maverick? $\longrightarrow$ stop

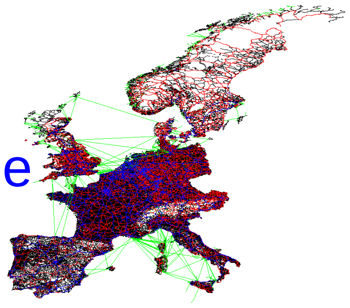
Result: superset of highway network

Local Queries (Highway Hierarchies Star, Europe)



Simple Solutions

Example: 10 000 × 10 000 table in Western Europe



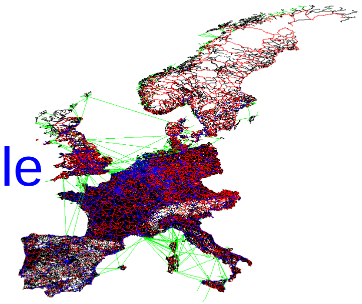
- apply SSSP algorithm (e.g. DIJKSTRA) $|S|$ times $\approx 10\,000 \times 10\,\mathrm{s} \approx$ one day

- apply P2P algorithm (e.g. highway hierarchies[1]) $|S| \times |T|$ times $\approx 10\,000^2 \times 1\,\mathrm{ms} \approx$ one day

---

[1] requires about 15 minutes preprocessing time

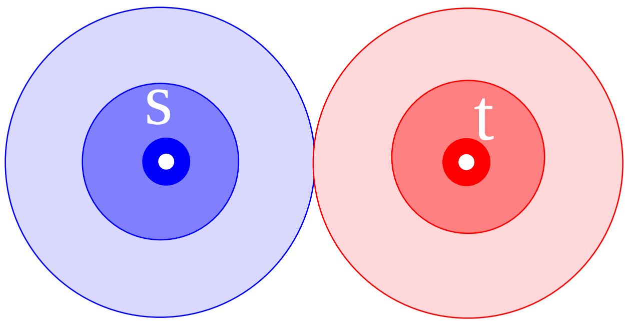
Our Solution



Example: 10 000 $\times$ 10 000 table in Western Europe

- many-to-many algorithm based on highway hierarchies[1]

$\approx$ one minute



[1] requires about 15 minutes preprocessing time

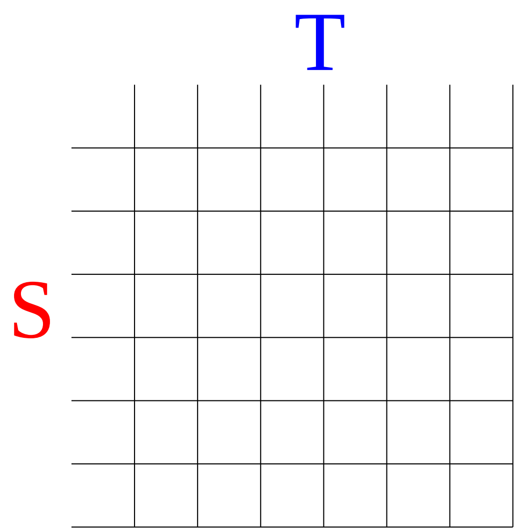
Main Idea

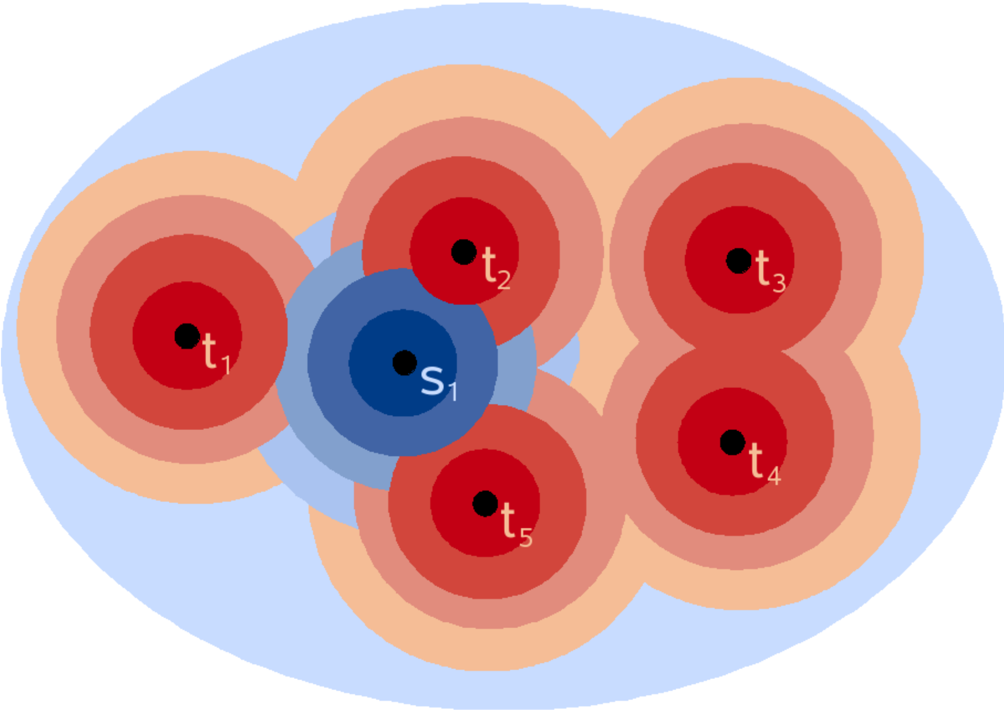
- instead of $|S| \times |T|$ bidirectional highway queries
- perform $|S| + |T|$ unidirectional highway queries

Algorithm

- maintain an $|S| \times |T|$ table D of tentative distances

  (initialize all entries to ∞)





- for each $t \in T$, perform backward search

  store search space entries $(t, u, d(u,t))$

- arrange search spaces: create a bucket for each u

- for each $s \in S$, perform forward search

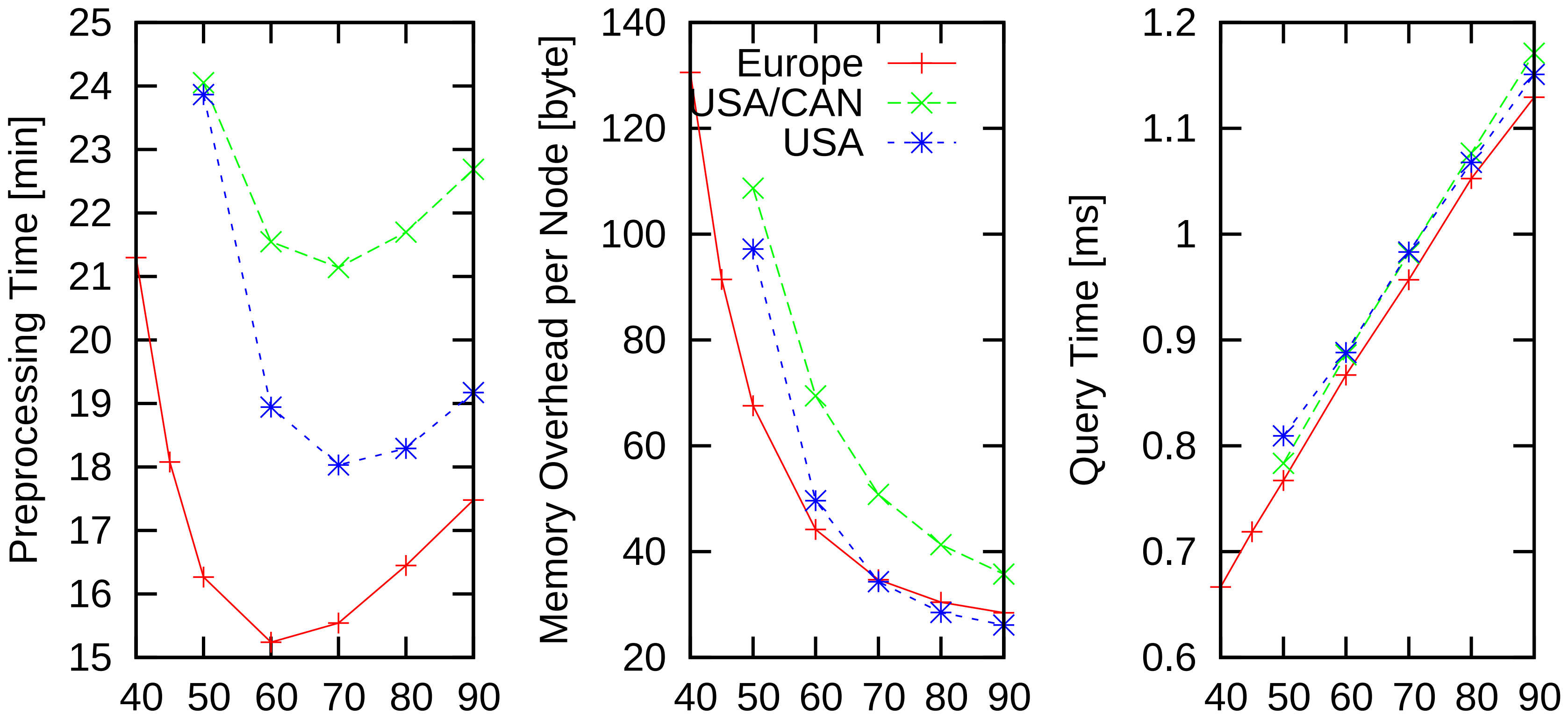
  at each node u, scan all entries $(t, u, d(u,t))$ and

  compute $d(s,u) + d(u,t)$, update $D[s,t]$

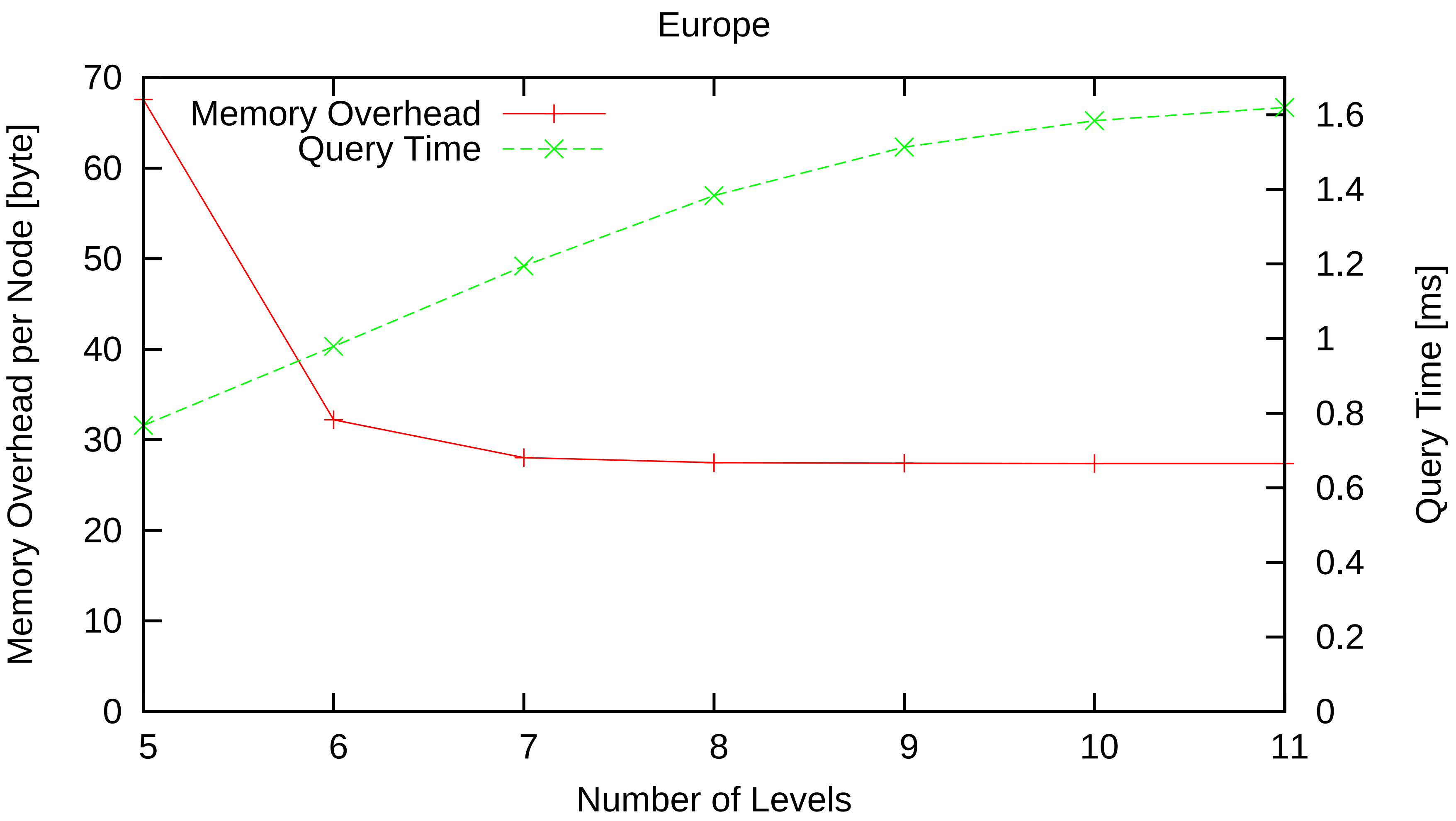
Different Combinations

		Europe				
metric		∅	DistTab	ALT	both	
time	preproc. time [min]	17	19	20	22	
	total disk space [MB]	886	1 273	1 326	1 714	
	#settled nodes	1 662	916	916	686	(176)
	query time [ms]	1.16	0.65	0.80	0.55	(0.18)
dist	preproc. time [min]	47	47	50	49	
	total disk space [MB]	894	1 506	1 337	1 948	
	#settled nodes	10 284	5 067	3 347	2 138	(177)
	query time [ms]	8.21	4.89	3.16	1.95	(0.25)

Neighbourhood Size



Number of Levels



Contraction Rate